AN OPERATOR-THEORETIC APPROACH FOR TRAFFIC PREDICTION

A Thesis
Presented to
The Academic Faculty

By

Esther P. Ling

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

AN OPERATOR-THEORETIC APPROACH FOR TRAFFIC PREDICTION

Approved by:

Dr. Sam Coogan, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mark Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 27, 2018

”Discovering the truth about ourselves is a lifetime’s work, but it’s worth the effort.”

Fred Rogers

For my family, who let me give up a perfectly good job in the supposed pursuit of knowledge.

ACKNOWLEDGEMENTS

This work began to begin when I was introduced to dynamic mode decomposition and Koopman Operator theory by my advisor, Sam Coogan. I'm thankful to him for introducing a set of theory that would have otherwise escaped my notice, and for providing constant feedback and directional prodding along the way. I would also like to acknowledge Lillian Ratliff and Liyuan Zheng for insightful discussions as collaborators in this project. Also, I would like to thank my committee members, Magnus Egerstedt and Mark Davenport, for their helpful feedback.

Special thanks is also due to Jorge Laval, who provided helpful discussions on traffic, and Byron Boots, who unwittingly extended my analysis into the theory of learning dynamical systems.

Finally, I must mention Alberto, Max and Mohit, for keeping things merry in the lab.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Abbreviations

CDMD Companion Dynamic Mode Decomposition

DMD Dynamic Mode Decomposition

DMDc Dynamic Mode Decomposition with control

FFT Fast Fourier Transform

HDMD Hankel formulation of Dynamic Mode Decomposition

HDMDc Hankel formulation of Dynamic Mode Decomposition with control

SDMD Standard Dynamic Mode Decomposition

SDMDc Standard Dynamic Mode Decomposition with control

SVD Singular value decomposition

Variables and Symbols

$*$ conjugate-transpose

Δt binning interval

δt cycle time

$\Im$ imaginary part

Λ diagonal matrix of eigenvalues

λ	eigenvalue
$\mathbb{C}$	set of complex numbers
$\mathbb{R}$	set of real numbers
$\mathbb{U}$	input matrix for DMDc
$\mathcal{F}$	space of observables
$\mathcal{K}$	finite-dimensional representation of Koopman Operator
$\mathcal{P}$	projection operator mapping $\mathcal{F}^M \rightarrow \mathcal{K}_N$
$\mathcal{T}$	nonlinear law mapping $\mathcal{X} \rightarrow \mathcal{X}$
$\mathcal{U}$	infinite-dimensional Koopman Operator
$\mathcal{X}$	state space
Ω	augmented data matrix
ϕ	Koopman eigenfunction
Ψ	Koopman modes stacked column-wise
ψ	Koopman mode
Σ	diagonal matrix containing singular values from the SVD of a matrix
A	linear operator or $M \times M$ matrix in DMD
C	companion matrix
D	number of variables in x
f	observable or output function mapping states to measurements, $f \in \mathcal{F}$
G	augmented operator

h	number of time-delayed rows added to a data or input matrix
K	matrix realization of $\mathcal{K}$
k	time-step
M	number of variables in y
N	number of snapshots or observations
r	rank-truncation in the SVD
T	prediction horizon
U	left singular vectors from the SVD of a matrix
u	control input, $u \in \mathbb{R}^Q$
V	right singular vectors from the SVD of a matrix
W	matrix of eigenvectors stacked column-wise
x	state, $x \in \mathcal{R}^D$ belonging to state space $\mathcal{X}$
X_1	data matrix
X_2	data matrix, time-shifted version of X_1
y	observation, $y \in \mathcal{R}^M$

SUMMARY

In this thesis, we develop a case for data-driven modeling of traffic flow at signalized intersections using an operator-theoretic framework. Traffic at signalized arterials is known to be problematic to model, due to significant short-term fluctuations and recurring unstable conditions. Moreover, the collection of large volumes of traffic data from traffic sensors begs the use of creating data-informed models. We highlight the Koopman Operator as a unifying framework and elaborate on existing data-driven algorithms that have been connected to it. We divide these algorithms into two broad classes: dynamic mode decomposition and the Arnoldi method. This thesis focuses mainly on dynamic mode decomposition. First, we demonstrate the performance of dynamic mode decomposition in learning dynamics of the highly oscillatory signalized traffic data. We show the rank-deficiency limitation, and how to overcome it using time-shifted observations. Additionally, we show that modeling signal phases as exogeneous control input performs better compared to solely using time-shifted observations. Second, we apply dynamic mode decomposition to the inverse problem of signal and phase timing recovery from vehicular flows. Third, we develop an algorithm for real-time instability detection that is able to distinguish between queue growth arising from a traffic accident and regular peak hours. Finally, we propose a method for simulating a modified signal phase for queue-breakdown mitigation, using dynamic mode decomposition with control.

CHAPTER 1
INTRODUCTION

1.1 From Freeways to Signalized Arterials

With the advent of aggregated traffic data collection and socio-economical benefits of relieving traffic congestion, studies on characterizing and understanding traffic behavior remain highly relevant. For example, real-time traffic prediction leads to improved advanced traveler information systems (ATIS) and route optimization. However, while many studies have been carried out in traffic prediction, most of these have been directed at urban freeway traffic.

Moreover, according to [1], signalized traffic exhibits phenomena such as large short-term fluctuations induced by traffic signaling and unstable conditions arising from congestion which are not found in freeway traffic. [2] reports that higher oscillations in arterials cause accuracy of their model to decrease by 10-20%. This suggests room for developing new approaches in the context of signalized traffic. Additionally, the questions of interest shift towards identifying congestion and injecting control into the system to minimize unstable conditions.

1.2 Data-Driven Modeling

Much of previous work on traffic prediction have one thing in common: reliance on equations to include prior beliefs into the model. For example, for modeling traffic flow in freeway data, [2] defined a space-time threshold autoregressive equation to capture transient behavior in their model. Comparatively, [3] used a multivariate spatial-temporal autoregressive model to account for transient behavior. The Kalman filter proposed by [1] to predict future traffic at urban signalized arterials relies on correct implementation of the

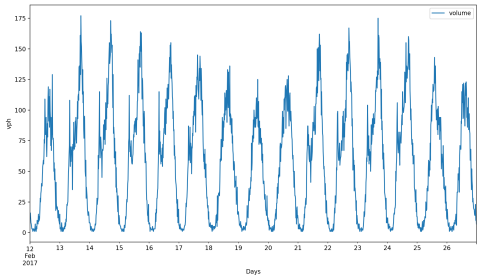


Figure 1.1: Traffic flow profile over several days (vehicles per hour) at 15 min resolution.

correct motion and observation equations. While these are examples of solid approaches, we wanted to develop a framework that is compelled by data foremost and less reliant on equation-based specifications. Do any such methods exist? We elaborate in the next section, by first setting up the notion that traffic is nonlinear.

1.3 Traffic as a Nonlinear System

Traffic is highly nonlinear (see Figure 1.1). Kamarianakis, Gao and Prastacos write in [4], "*the intensely oscillating behavior of traffic variables, reflected as frequent and sudden shifts to extreme values, lead investigators to assume non-stationarity and nonlinearity*." Even traffic data aggregated at lower resolution of 15 minutes shows nonlinearity, attested to in [4], as evidenced by time-series models having multiple regimes. One possible treatment thus is to model traffic as a nonlinear dynamical system.

Nonlinear systems are traditionally analyzed through the eye-lens of state evolution however, this shows limitations as the number of states grow exponentially large [5]. This is arguably the case in a traffic network, due to the spatial dependency of the system. To illustrate, sensor locations or flows at turn-movements for each intersection, could each represent a state variable, and this number increases as the size of the network in consideration increases. Even if prediction is tested locally, in order to scale well during implementation, a framework is needed for handling high-dimensional systems.

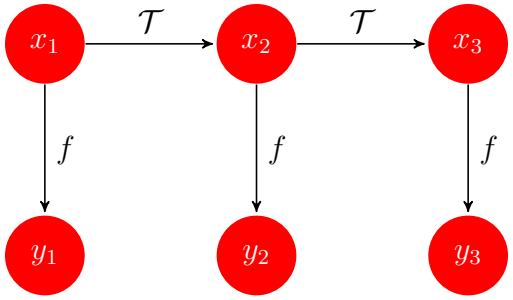


Figure 1.2: A typical Markov Chain depicting the evolution of states, where x_i are the states and y_i are observations of the state.

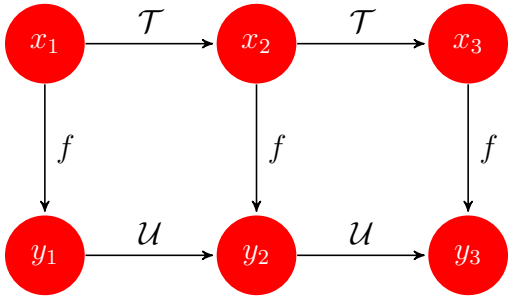


Figure 1.3: In the Koopman Operator framework, explicit dependencies are added between y_i to depict evolution of functions on states.

1.4 Koopman Operator: A Unifying Framework

In 1931, Koopman [6] postulated an infinite-dimensional linear operator that acts on the space of all possible observable functions on the state. He builds on the idea of linear transformations in a Hilbert Space. Intuitively, instead of trajectories on a manifold that evolve according to a nonlinear law, in the transformed coordinates, the observables evolve according to a linear law.

More recently, in 2012, Budisic et. al [5] highlight that the Koopman Operator presents an alternate view to the "dynamics of states" framework, namely, the "dynamics of observables" framework. See Figure 1.2 and Figure 1.3 to compare the two frameworks. Studying evolution of functions on the state space has opened new avenues for problem solving, such as (1) not requiring an analytical model of the system [7], (2) providing notions of modal analysis, and (3) is superior to linearization techniques in that it is able to capture the full nonlinear dynamics of the system [5]. This will be discussed further in following Chapters.

It is evident that if we are trying to approximate an infinite-dimensional operator, our best guess can only be a finite-dimensional one. It turns out that existing data-driven algorithms have been connected to approximating the spectrum of the Koopman Operator, namely, dynamic mode decomposition (DMD) and the Arnoldi algorithm. See [8] for connections between dynamic mode decomposition and Koopman mode approximation. More recently, in 2017, Arbabi and Mezic [9] showed how these algorithms converge to the Koopman Modes, under assumptions that the dynamical system is ergodic, only discrete spectra exist, and existence of an invariant finite-dimensional Hilbert subspace that contains the observables that are measured.

Broadly, these two classes of algorithms can be thought of as searching for different finite-dimensional subspaces on which to project the modes of the Koopman Operator. Arnoldi type-algorithms [10] borrow from Krylov subspaces theory and formulate a companion matrix, while dynamic mode decomposition [11], [12], [13] searches for a best fit linear operator. Extensions to each of these algorithms have been developed, for example, Vector Prony algorithm [7], which appends previous time-step information into the measurement vector, and Extended DMD [13], [14] which appends nonlinear mappings of the mesaurements to the measurement vector. The algorithms and their extensions are charted in Figure 1.4.

The takeaway from this section is as follows. We started out with the goal of finding an equation-free method to drive modeling traffic flow. We established traffic dynamics as a nonlinear system and saw how the Koopman Operator is well-suited for modeling nonlinear dynamical systems. In the following Chapters, we will see how the data driven algorithms connected to Koopman Operator theory (dynamic mode decomposition and Arnoldi method) lend us notions of modal decomposition that is well suited for handling urban signalized traffic data.

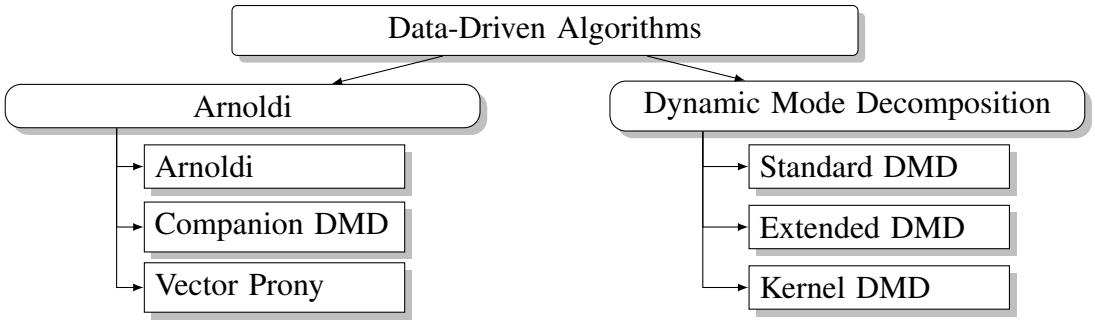


Figure 1.4: Data Driven Algorithms for approximating the Koopman Operator

1.5 Contributions of Thesis

The main theme of this thesis is to develop a case for a data-driven modeling of traffic flow at signalized intersections, zoning in on the Koopman Operator Theory and its realized algorithms that provide a framework for prediction, control, understanding spatio-temporal relationships and instability occurrences.

The remainder of the thesis is outlined as follows. In Chapter 2, we explore the theory behind Koopman Operator and the two main classes of its realized data-driven algorithms: Dynamic Mode Decomposition and the Arnoldi method. In Chapter 3, we elaborate more on our case study location and the data available. In Chapter 4, we investigate the performance of Koopman Operator framework in learning traffic dynamics, and in particular, compare various DMD algorithms on *real data*. In Chapter 5, we demonstrate usefulness of DMD to analyze spatio-temporal dependencies in the traffic network, particularly, for signal timing and phase recovery. In Chapter 6, we develop an algorithm for detecting instability in the network, and propose a method for control during an unstable condition. In Chapter 7, we provide concluding thoughts and suggestions for future work.

CHAPTER 2
THEORY AND RELATED WORK

2.1 Koopman Operator Theory

This section provides some theoretical background for Koopman Operator theory, of which we credit the proofs developed here to [7], and also [5], who provides a comprehensive review on Koopman Operator theory as a whole. For a more thorough exposition on spectra theory, refer to [15].

2.1.1 Definition

Here, we define the Koopman Operator. We start by making the transition from a nonlinear, finite-dimensional dynamical system to a linear, but infinite-dimensional operator. Consider a discrete-time nonlinear dynamical system that evolves on a finite D-dimensional manifold $\mathcal{X}$:

$$\begin{aligned} x_{k+1} &= \mathcal{T}(x_k) \\ y_k &= f(x_k) \end{aligned} \tag{2.1}$$

where $x_k \in R^D$ is a state vector that lives in state space $\mathcal{X}$, $\mathcal{T}$ is a nonlinear vector map that describes the evolution of state trajectories, $y_k \in C^M$ is the measured output, and $f \in \mathcal{F}$ is the output function or observation of the state that maps states to measurements: $f : \mathcal{X} \to \mathbb{C}^M$, which we will also refer to as an observable, and k is the time-step.

Now, we shift our paradigm from evolution of states $\mathcal{T}(x_k)$ to evolution of observables $f(x_k)$ on the states. We ask the following question: is there a linear operator that can act on $f(x_k)$, instead of the traditional $\mathcal{T}$ acting on x_k, such that it advances the observed measurements at each step? See again Figures 1.2 and 1.3 to compare the difference in

frameworks. This was Koopman's original idea in [6]. Here is where we introduce the Koopman Operator $\mathcal{U}$, showing how it acts on f:

$$\mathcal{U}f := f \circ \mathcal{T} \tag{2.2}$$

To defend the notion that $\mathcal{U}$ is linear, we provide the following proof [7]. Given any two $f_1, f_2 \in \mathcal{F}$, and $\alpha_1, \alpha_2 \in \mathbb{C}$, then

$$\mathcal{U}(\alpha_1 f_1 + \alpha_2 f_2) = (\alpha_1 f_1 + \alpha_2 f_2) \circ \mathcal{T} = \alpha_1(f_1 \circ \mathcal{T}) + \alpha_2(f_2 \circ \mathcal{T}) = \alpha_1(\mathcal{U}f_1) + \alpha_2(\mathcal{U}f_2)$$

As noted in [16], using $\mathcal{U}$ trades finite-dimensional dynamics for infinite-dimensional dynamics, since it evolves on an infinite-dimensional function space. Also, [7] point out that $\mathcal{U}$ captures the full original system dynamics and does not rely on linearization techniques, which brings extensions to control techniques, such as explored in [16].

2.1.2 Spectral Decomposition

The choice of the function space $\mathcal{F}$ is crucial to the analysis of the Koopman Operator spectrum, but we will not dive into the details here. Instead, refer to [15] and [5]. In short, the $\mathcal{L}_2$ space is normally chosen, which has connections to ergodic theory and measure-preserving dynamical systems. All this is to say that, in this case, $\mathcal{U}$ is unitary, i.e. has eigenvalues on the unit circle, given that the dynamics lie on an attractor [5].

A unitary $\mathcal{U}$ decomposes into a singular $\mathcal{U}_s$ and regular $\mathcal{U}_r$ part, which are connected to the discrete and continuous spectrum respectively. Then,

$$\mathcal{U} = \mathcal{U}_s + \mathcal{U}_r = \sum_{j=1}^{\infty} \lambda_j \phi_j \psi_j + \mathcal{U}_r \tag{2.3}$$

where $\psi_j \in \mathbb{C}^M$ is the Koopman Mode corresponding to the Koopman Eigenvalue λ_j, that belongs to a particular observable f, $\phi_j \in \mathcal{F}$ is the Koopman eigenfunction, and we refrain

from defining $\mathcal{U}_r$, as we will soon only focus on the discrete spectrum.

Now we can express the observables $f \in \mathcal{F}$ in terms of the spectral decomposition of $\mathcal{U}$:

$$f(x_k) = (\mathcal{U}^k f)x_0 = \sum_{j=1}^{\infty} \lambda_j^k \phi_j(x_0)\psi_j + \textit{contribution from } \mathcal{U}_r \tag{2.4}$$

Suppose that f is in the span of the eigenfunctions ϕ_j, then, f can be written as a basis expansion, where the Koopman Modes ψ_j contain the expansion coefficients:

$$f = \sum_{j=1}^{\infty} \psi_j \phi_j \tag{2.5}$$

With abuse of notation, when indexing into one spatial dimension i, we can write:

$$f(i) = \sum_{j=1}^{\infty} \psi_j(i)\phi_j = \psi(i)^T \phi,\ i = \{1, ..., M\} \tag{2.6}$$

Under this assumption that $f \in \textit{span}\ \{\phi_j\}$, we drop the $\mathcal{U}_r$ dependency and write $f(x_k)$ in terms of the discrete spectrum only:

$$f(x_k) = \sum_{j=1}^{\infty} \phi_j(x_k)\psi_j = \sum_{j=1}^{\infty} \lambda_j^k \phi_j(x_0)\psi_j \tag{2.7}$$

And again, indexing into one spatial dimension:

$$f(x_k(i)) = \sum_{j=1}^{\infty} \phi_j(x_k(i))\psi_j = [\phi(x_k(i))]^T \psi,\ i = \{1, ..., M\} \tag{2.8}$$

Interpretively, the Koopman Eigenvalue λ_j describes the temporal behaviour of its corresponding Koopman Mode ψ_j, where the growth or decay rate is determined by $|\lambda_j|$ and the frequency of oscillation comes from $\angle\lambda_j$ [7]. Moreover, since the Koopman Mode contain the expansion coefficients (Equation 2.7), we can think of $|\psi_j(i)|$ representing the amount of influence a variable i exerts in that temporal frequency. On the other hand, $\angle\psi_j(i)$ describes the relative phase of oscillation [17].

2.1.3 Data-Driven Algorithms

Until now, we have been considering an infinite number of expansion terms, given that the Koopman Operator $\mathcal{U}$ is infinite-dimensional, and hence has $\{j = 1, 2, ..., \infty\}$ spectral components showing up in the expansion of Equation 2.7. However, when moving to data-driven applications, one needs a *finite* number of terms, after all, by definition, having a data-centric algorithm implies finiteness of data. What data-driven algorithms do then, is to find a finite-dimensional subspace on which to approximate $\mathcal{U}$ [5].

Suppose a system evolving in time has been sampled at time $k = \{1, ..., N\}$ at M locations, i.e. measurement data of $\{y_k\}_{k=1}^N$ is available, where $y_k \in \mathbb{R}^M$. The observables are then $f \in \mathcal{F}^M$ and the function space spanned by the measured observables is $\mathcal{K}_N :=$span$\{\mathcal{U}^k f\}_{k=1}^N$, where $\{\mathcal{U}^k f\}_{k=1}^N$ is assumed to be a linearly independent set.

Define a projection operator $\mathcal{P}_N : \mathcal{F}^M \to \mathcal{K}_N$. Then, we have the following operator $\mathcal{P}_N\mathcal{U} : \mathcal{K}_N \to \mathcal{K}_N$, which can be written in matrix form as $K : \mathbb{C}^N \to \mathbb{C}^N$. With this formulation, finding the Koopman Operator, $\mathcal{U}$ has been recast as finding the best finite-dimensional matrix K from the data [7].

Then, we relate the observables acting on the state data $f(x_k)$ in terms of the measured data y_k:

$$f(x_k) = y_k = \sum_{j=1}^{N} \psi_j \phi_j(x_k) \tag{2.9}$$

In [9] it is shown that as $N \to \infty$, then the approximated Koopman Modes and Eigenvalues (spectra) approach that of the actual $\mathcal{U}$.

Incidentally, there are several algorithms that have been developed independently of the Koopman Operator, but now have been connected to approximating $\mathcal{U}$. For example, [8] made connections with dynamic mode decomposition (DMD), while [5] made connections to the Arnoldi algorithm. These algorithms are equation-free, in that they don't make prior assumptions on $\mathcal{T}$ and $\mathcal{F}$ [7]. Instead, they rely on taking snapshots of the data in time.

Armed with this new formulation, a high level view of these algorithms is that they start

Algorithm 1 Dynamic Mode Decomposition

Formulation: $X_2 = \mathrm{A}X_1$

1: Compute the SVD of X_1:
$X_1 = U\Sigma V^*$,
where $U \in \mathbb{C}^{M\times r}$, $\Sigma \in \mathbb{C}^{r\times r}$, $V \in \mathbb{C}^{N\times r}$ and r is the rank truncation.
2: Project A onto the POD/PCA modes U:
$\hat{A} = U^*AU = U^*X_2V\Sigma^{-1}$, where A = $X_1^\dagger X_2 = V\Sigma^{-1}U^*$
3: Obtain the DMD eigenvalues from the eigen-decomposition of $\hat{A}$:
$\hat{A} = W\Lambda W^T$
4: Compute the DMD modes:
$\Psi = X_2V\Sigma^{-1}W$

by forming two time-shifted data matrices, X_1 and X_2:

$$X_1 = \begin{bmatrix} x_1 & \dots & x_{N-1} \end{bmatrix} X_2 = \begin{bmatrix} x_2 & \dots & x_N \end{bmatrix} \tag{2.10}$$

where $x_i \in \mathbb{R}^M$, M is number of states and N is number of snapshots.

In the following sections, we explain each of the algorithms in more detail.

2.2 Dynamic Mode Decomposition

2.2.1 Historical Origins

The first algorithm we consider is dynamic mode decomposition (DMD). In 2008, Schmid and Sesterhenn [18], and later in 2010 [11] developed DMD in the context of extracting coherent structures in high-dimensional fluid data. Given its historical origins, DMD typically arises in cases when the spatial dimension is greater than the number of observations, i.e. when X_1 and X_2 are tall and thin matrices. Refer to Algorithm 1 for the Standard DMD algorithm.

2.2.2 Mathematical Setup

In the previous section, we saw how the Koopman Operator is first of all linear, and secondly, acts on the space of functions of measurements. Our best attempt at realizing this

operator became a problem of finding a finite-dimensional matrix K.

$$\min_{A} ||X_2 - AX_1||_F^2 \tag{2.11}$$

where $A \in \mathbb{R}^{M \times M}$.

In dynamic mode decomposition, a best fit linear operator A that minimizes the squared Frobenius norm between the two time-shifted matrices as in Equation 2.10 is sought, as a solution to the optimization program in Equation 2.11. Here, A plays the role of the finite-dimensional matrix K. Again, Schmid [11] stresses that learning this matrix A from data snapshots imposes no prior assumptions on the model, emphasizing the equation-free trait of DMD.

2.2.3 Connections to Subspace Identification

Let us pull away briefly from the connections of DMD to the Koopman Operator in its approximation of the Koopman modes. Considering purely its mathematical setup in Equation 2.11, DMD seeks a best fit linear operator that minimizes the squared loss between two time-shifted data matrices. This problem in itself can be connected to subspace identification method [19], which uses the same formulation. For instance, see [20] for an instantiation of the same optimization program. (There is arguably a larger school of thought that both DMD and subspace identification fall under, which is algorithmic modeling. Rapp et. al [21] describe the evolution of models that arose from applied mathematics in dynamical systems; with Newtonian physical modeling shifting to the Poincaré geometric paradigm, and more recently, algorithmic modeling, which relies on data collected at intervals).

Where DMD and subspace identification veer off however, is in the focus of the problem. In the control community, subspace identification has been largely explored in the context of increasing accuracy of its one-step or multi-step prediction [22], [23], and the efficiency of the computation [20], [24]. On the other hand, DMD has been connected more with its ability to decompose complex systems into individual spatio-temporal modes

oscillating at different frequencies. (Nevertheless, we should not also exclude its other usefulness in prediction and reduced order modeling, which in fact is what we explore in Chapter 4). It is helpful to keep this in mind; though being mathematically equivalent, they bring different connotations to mind when it comes to applications.

2.2.4 Proper Orthogonal Decomposition and Fourier Transform

Are there any other connections? We can also think of DMD as providing a nice combination [25] of proper orthogonal decomposition (POD) [26] in space and Fourier Transforms [27] in time. Proper orthogonal decomposition, also known as principal components analysis (PCA), is a dimensionality reduction technique [28] that relies on the singular value decomposition (SVD). DMD also employs the SVD, which encapsulates the dimensionality reduction aspect. See again Algorithm 1. However, in DMD, the data matrix contains snapshots evolving in time, which introduces the notion of dynamics. In this case, U contains spatial information, while V contains temporal correlations. (Note that in POD, if the data does not contain temporal information, then V does not have the same interpretation of being temporally ingrained [25]). Additionally, the DMD algorithm extracts eigenvalues which can have a growth or decay component that may oscillate at certain frequencies, if they are complex-valued. These frequencies can be compared against those extracted by Fourier analysis, thus providing the POD and Fourier combination.

2.2.5 Increasing Spatial Dimension

There is a cautionary note to be pointed out when using DMD in the context of prediction. One immediate limitation is when the number of states is smaller than the number of observations. Tu et. al [12] observe that when the dynamics is dominated by oscillations, DMD fails to capture the oscillatory eigenvalues, and thus the reconstructed solution does not contain the periodicity that one would expect to see propagated. They demonstrate this using an example of a standing wave. Also, [25] call out two limitations of DMD in

Algorithm 2 DMD with Control

1: Compute the SVD of the augmented data matrix Ω:
$\Omega = \tilde{U}\tilde{\Sigma}\tilde{V}^*$
2: Compute the SVD of X_2:
$X_2 = \hat{U}\hat{\Sigma}\hat{V}^*$
3: Compute the approximation of the augmented operator G:
i) Extract A from G:
$A = \hat{U}^* X_1 \tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_1^*\hat{U}$
where $\tilde{U}_1$ is formed from the first M rows of $\tilde{U}$.
ii) Extract B from G:
$B = \hat{U}^* X_1 \tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_2^*$
where $\tilde{U}_2$ is formed from the last Q rows of $\tilde{U}$.
4: Perform eigen-decomposition of A:
$\tilde{A}W = W\Lambda$
5: Compute the dynamic modes of A:
$\Psi = X_1 \tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_1^*\hat{U}W$

handling translational and rotational invariances, which comes from the limitation of the SVD. In a complex nonlinear system, one cannot expect that using a few number of spatial dimensions would be sufficient for modeling and predicting future states. In fact, this should not trouble us, given that DMD originated in the context of characterizing high-dimensional fluid flows. However, this is not to exclude using DMD when we have a small state dimension, as there are cases in which it still works, and there are ways to circumvent the issue.

In the case when DMD "does not work", how should we deal with a low spatial dimension? In short, we enrich the spatial column. This would be natural to the machine learning community, where increasing the size of the feature vector often results in better learning. Since we are dealing with a dynamical system, the first way in which we can do so is by appending time-delayed observations into the feature vector. Then, in Equation 2.11, the X_1 and X_2 matrices become Hankel matrices. This is also mentioned in [12] as a solution to the standing wave example, which connects the appending of time-delayed observations with the eigensystem realization algorithm (ERA) that is found in subspace identification literature.

However, embedding future states into the columns (1) assumes we have enough data, and (2) requires some work in tuning the number of rows added. An alternative method

to increase the spatial dimension is to add nonlinear feature mappings of the observations. This is first proposed by [13], who suggests choosing a dictionary of functions to be applied onto the observations. Their motivation stems from pursuing a better approximation of the Koopman Operator; since the Koopman eigenfunctions span a subspace, choosing a "richer set of basis functions instead of relying on the original set of observations should lead to a better approximation. They call this method Extended Dynamic Mode Decomposition. And of course, if one considers nonlinear mappings, one must think of computational complexity and leveraging kernel methods, which is in fact what the same authors later propose in [14], known as Kernel Dynamic Mode Decomposition.

2.2.6 Adding Control

DMD can also be extended to include an exogenous control input [29], known as dynamic mode decomposition with control (DMDc). It considers the case of a dynamical system with input:

$$x_{k+1} = Ax_k + Bu_k \tag{2.12}$$

where $u_k \in \mathbb{R}^Q$ is the control action. We can then augment the matrices:

$$\begin{aligned} X_2 &= AX_1 + B\mathbb{U} \\ &= \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X_1 \\ \mathbb{U} \end{bmatrix} \\ &= G\Omega \end{aligned} \tag{2.13}$$

where $\mathbb{U}$ is formed by stacking the input observations:

$$\mathbb{U} = \begin{bmatrix} u_1 & \dots & u_N \end{bmatrix}$$

Now, a B matrix needs to be learned in addition to A. The DMDc algorithm is described

Algorithm 3 Arnoldi

Formulation: $X_2 = X_1 C$

1: Compute the coefficient vector c:
$c = X_1^{\dagger} x_N$, where x_N is the last snapshot.

2: Form the companion matrix C:
$$C = \begin{bmatrix} 0 & 0 & \dots & 0 & c_1 \\ 1 & 0 & \dots & 0 & c_2 \\ 0 & 1 & \dots & 0 & c_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_N \end{bmatrix}$$

3: Compute the eigenvalues of C:
$C = W \Lambda W^T$, Λ are called the Ritz values.

4: Form the Vandermonde Matrix, $\mathbb{T}$ using Λ:
$$\mathbb{T} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{N-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_N & \lambda_N^2 & \cdots & \lambda_N^{N-1} \end{bmatrix}$$

5: Compute the empirical Ritz vectors, which are the columns of V:
$V = X_1 \mathbb{T}^{-1}$

in full in Algorithm 2.

2.3 Arnoldi Algorithm

One critique of DMD is in its performance of handling problems with low spatial dimension as mentioned in [7] and [30]. [30] in particular argue that Standard DMD suffers from rank-deficiency and propose the companion matrix as the workaround, running experiments comparing the Arnoldi, Vector Prony and Standard DMD algorithms in their ability to recover frequencies close to the true ones as identified by a Fast Fourier Transform (FFT). While we have covered in Section 2.2.5 how a rank-deficient formulation of DMD can be remedied by stacking time-shifted observations, we provide this school of thought for completeness.

Algorithm 4 Companion DMD

Formulation: $X_2 = X_1\mathrm{C}$

1: Compute the coefficient vector c:
$c = X_1^{\dagger} x_N$, where x_N is the last snapshot.
2: Form the companion matrix C, similar to the Arnoldi algorithm.
3: Compute the eigen-decomposition of C:
$C = W\Lambda W^T$
4: Compute the modes:
$\Psi = X_1 W$

2.3.1 Mathematical Setup

In the Arnoldi algorithm, the optimization program can be thought of finding the best companion matrix C that minimizes:

$$\min_{C} ||X_2 - X_1 C||_F^2 \tag{2.14}$$

where $C \in \mathbb{R}^{(N-1)\times(N-1)}$ is a companion matrix.

It would be helpful to compare this formulation with the one in Equation 2.11. The first thing to note is that in Equation 2.11, A is of dimension $M \times M$, while here C is of dimension $(N-1) \times (N-1)$, where N is the number of snapshots and M is the spatial dimension. In the low spatial dimension case, i.e. when there are more observations than spatial dimension, notice that C is a larger matrix compared to A, and hence is better off in terms of rank. We describe the full Arnoldi algorithm in Algorithm 3.

2.3.2 Companion DMD

Companion DMD was first proposed in [9], which is a spin-off the Arnoldi algorithm. It also sets up the formulation as in Equation 2.14, but stops at the computation of C. The Koopman modes are approximated from C, instead of the Vandermonde Matrix $\mathbb{T}$ that appears in Algorithm 3. Companion DMD is described in full in Algorithm 4.

Algorithm 5 Vector-Prony

Formulation: $X_2 = X_1 C$

1: Form the Hankel matrices of delay-embedded snapshots:

$$\hat{X}_1 = \begin{bmatrix} x_1 & x_2 & \dots & x_{N-1} \\ x_2 & x_3 & \dots & x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_k & x_{k+1} & \dots & x_{2N-k} \end{bmatrix}, \hat{X}_2 = \begin{bmatrix} x_2 & x_3 & \dots & x_N \\ x_3 & x_4 & \dots & x_{N+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k+1} & x_{k+2} & \dots & x_{2N-k+1} \end{bmatrix}$$

where k is the number of snapshots to be embedded. Then, the following steps are identical to the Arnoldi algorithm.

2: Compute the coefficient vector c:
$c = \hat{X}_1^{\dagger} x_N$, where x_N is the last snapshot from the Hankel Matrix $\hat{X}_2$.

3: Form the companion matrix C, similar to the Arnoldi algorithm.

4: Compute the eigenvalues of C:
$C = W \Lambda W^T$, where Λ are called the Prony values.

5: Form the Vandermonde Matrix, $\mathbb{T}$ using Λ:

$$\mathbb{T} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{N-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_N & \lambda_N^2 & \cdots & \lambda_N^{N-1} \end{bmatrix}$$

6: Compute the Prony values, which are the columns of V:
$V = X_1 \mathbb{T}^{-1}$

2.3.3 Vector Prony

Vector Prony is to Arnoldi what the Hankel matrix is to DMD. In Vector Prony, time delayed measurements are added to the data matrices X_1 and X_2. The following steps are then the same, with computation of the Koopman Modes from the Vandermonde matrix. We provide the Vector Prony algorithm in Algorithm 5.

2.4 Review of Applications

We now provide a review of applications of Koopman Operator theory in various domains.

First, Koopman Operator theory has provided natural connections to reduced order modeling and modal decomposition. Consider again Equation 2.9. Suppose that the first $R \leq N$ variables can capture a sufficient amount of the system's dynamics. Then, we can represent the system dynamics using a reduced order model as in Equation 2.15, which has computational advantages. Some applications of this is in power systems [31], [7].

$$f(x_k) = \sum_{j=1}^{R} \lambda_j^k \phi_j(x_0) \psi_j \tag{2.15}$$

Limiting ourselves to summing the first R terms is not the only application. We could also group terms, based on frequency of oscillation (see Equation 2.16). This is called modal decomposition, i.e., partitioning of the system based on the frequency of the modes. Raak et. al [32] tested this methodology using the Arnoldi algorithm to partition a power systems network. In [33], Brunton et. al used DMD and divided their system into fast oscillating and slow oscillating modes when trying to separate foreground and background segments in a video, where they related the background with slow oscillating modes, and the foreground with fast oscillating modes.

$$f(x_k) = \sum_{j=1}^{R_1} \lambda_j^k \phi_j(x_0)\psi_j + \sum_{j=R_1}^{R_2} \lambda_j^k \phi_j(x_0)\psi_j + ... \tag{2.16}$$

The notion of growth and decaying modes sparked off ideas in quantitative finance and power systems. In [34], the presence of growth or decay modes were used to predict trading times. In [31], the number density of unstable Koopman eigenvalues was used to demonstrate short term stability detection as post analysis study of the 2006 European grid-wide disturbance.

Clustering of modes oscillating at similar frequencies also led to the idea of coherency analysis. In power systems again, swing dynamics is of importance; maintaining synchronous generators at the frequency is crucial to the stability of the system. [35] used Koopman modes to identify groups of generators oscillating at coherent frequencies. In a medical application, [17] used DMD to study how flu and measles spread.

Other than direct applications, further explorations to connect Koopman Operator theory with other fields are being studied. For instance, Brunton, Proctor and Kutz [36] have made connections with DMD and compressive sensing. A stochastic version of the Koopman Operator has also been considered [15], which then opens up connections to Hidden Markov Models (HMM). A Bayesian form of DMD has been explored in [37].

Applications to traffic are still limited to the best of our knowledge, with one study in-

volving freeway data [38], who compared dominant modes in morning and evening traffic as a means of differentiating the dynamics in those two periods. Drawing from the applications in other domains reviewed here, we next demonstrate three novel applications in the traffic domain in the following Chapters.

CHAPTER 3

DATA OVERVIEW

Our case study location is a set of arterials connected by signalized intersections in Montgomery County, Maryland. In total, there are 7 intersections in the network. In this Chapter, we explain the data used in the experiments in later Chapters, which may be helpful in understanding the problem formulations. It may be safely skipped at first reading, and returned to when following the experimental sections.

3.1 Dataset

The data comprises traffic sensor data over the span of two years (2015 to 2017), from a network of intersections in Montgomery County, Maryland. In total, there are 7 intersections in the network. See Figure 3.1. We code-name each intersection as follows:

1. I1: Montrose Pkwy & Towne Rd
2. I2: Montrose Rd & Towne Rd
3. I3: Montrose Pkwy & E Jefferson
4. I4: Montrose Rd & Montrose Pkwy
5. I6: Montrose Rd & E Jefferson
6. I7: Montrose Rd & Tildenwood
7. I8: Montrose Rd & Farm Haven

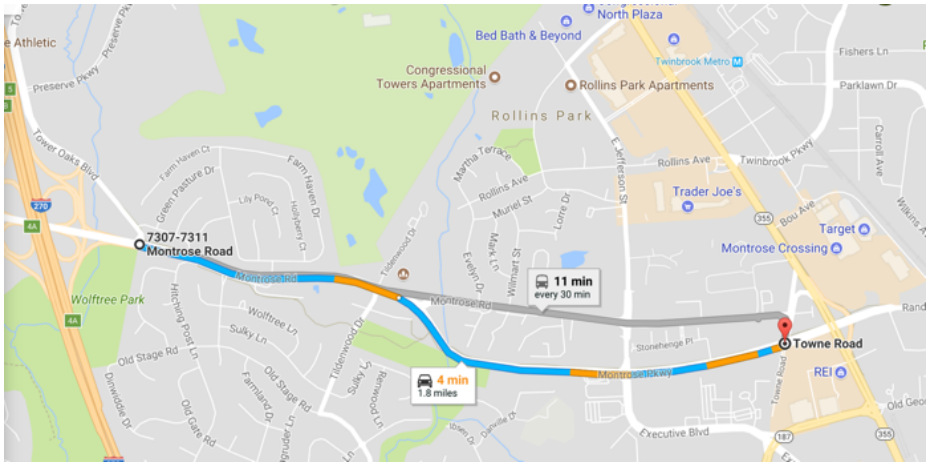


Figure 3.1: A Google Map view of the Network Location.

3.2 Topology of an Intersection

Within an intersection are up to 4 legs, corresponding to directions: Leg 1 is Southbound (SB), Leg 2 is Northbound (NB), Leg 3 is Westbound (WB), and Leg 4 is Eastbound (EB).

Within each leg are lanes. For this particular network, there are up to 5 lanes in any leg. Each lane has an associated turn-movement, which can be one of three: Left-Turn (LT), Through (T), and Right-Turn (RT). Note that a lane can have one or more turn-movements associated with it. For instance, a lane might have both a Left-Turn and Through movement. It is also possible that a particular leg might not have all turn-movements.

Each lane has an additional Type designation. There are two Lane-Types: Inbound and Outbound. The Inbound Lane-Type corresponds to lanes approaching the intersection, while the Outbound Lane-Type corresponds to lanes leaving the intersection. For a study involving vehicle flows and queue lengths approaching the traffic light, the Inbound Lane-Type will be the relevant one to use.

The above hierarchy of Intersection-Leg-Movement-Type results in a total of 147 individual lanes in this particular network. When only considering Inbound Lane-Type lanes, this number becomes 88.

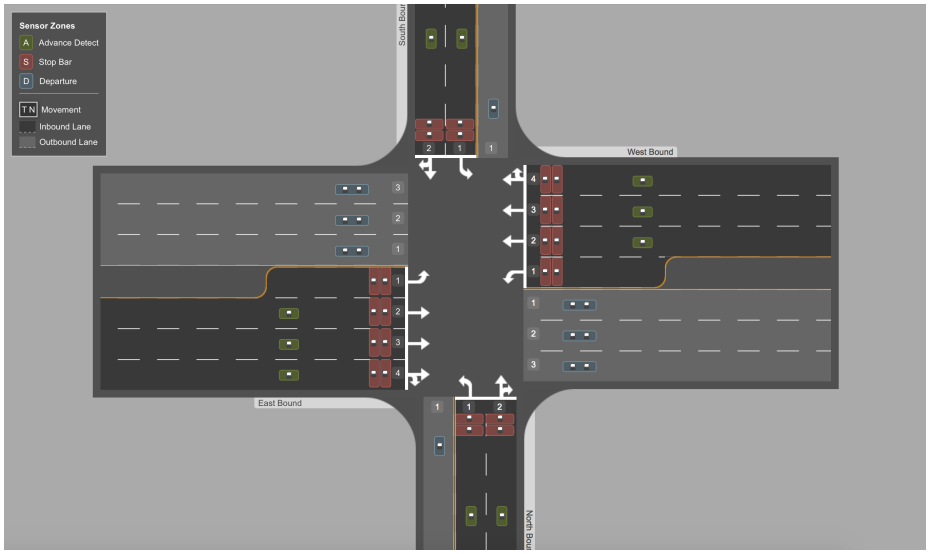


Figure 3.2: An example intersection (I7: Montrose Rd & Tildenwood) depicting Legs, Lanes, Stopbar Detectors (in red), Advanced Detectors (in green) and Departure Detectors (in blue) placements. Figure courtesy of Sensys Networks.

3.3 Sensors and Logging

Inbound Lane-Type lanes may have up to two sensors present: stopbar detectors and advanced detectors, while Outbound Lane-Type lanes only have departure detectors. For the Inbound Lane-Type lanes: Advanced detectors are positioned ahead of the traffic light to count vehicles entering the leg. Stopbar detectors on the other hand are positioned at the departure of the traffic light to count vehicles exiting the leg. Refer to 3.2 to see positioning of Advanced detector and Stopbar detector placements. For Outbound Lane-Type lanes, departure detectors are placed at the entrance of the leg to count vehicles leaving the intersection.

The data is logged as a time-stamped event every time a vehicle crosses either of the sensors described above. Other information captured with the logged event are: speed (mph), occupancy time (s), current phase (red / yellow / green), and time to end of phase (s).

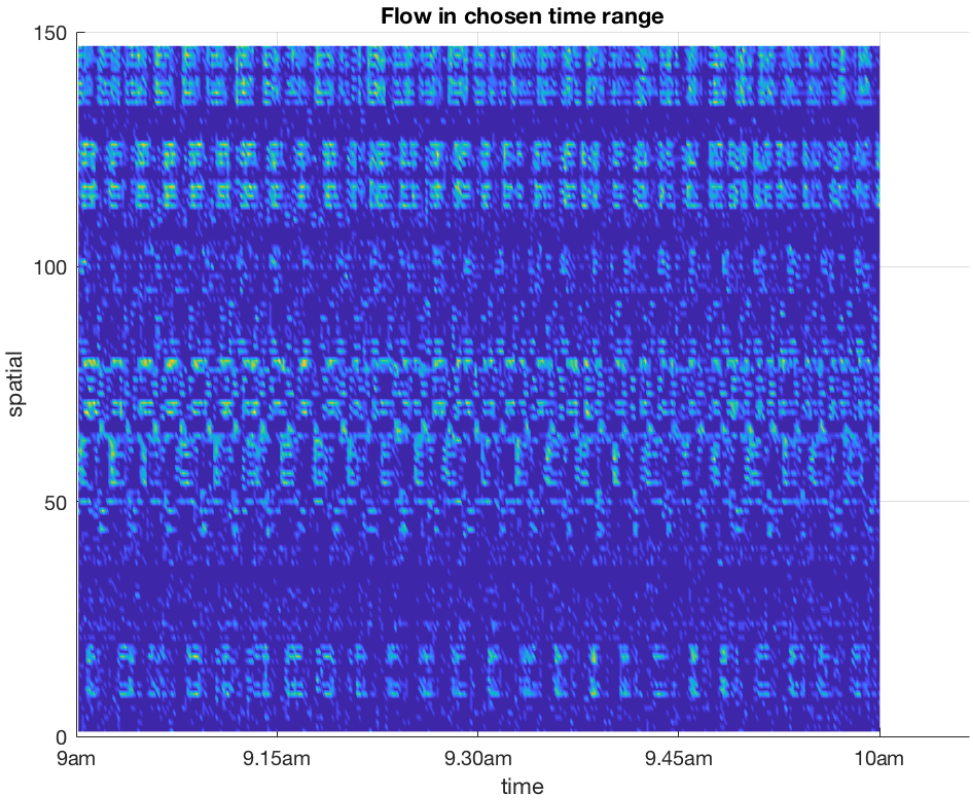


Figure 3.3: Flow (vehicles per hour) at I3, from 9am-10am on 14 Feb 2017

3.4 Suggested Use of Data

We suggest two manners in which to process the raw data:

1. **Vehicle Flow**

   To obtain vehicle flow, the total number of sensor events can be counted and aggregated into bins. The flow per hour (vph) can be obtained by multiplying the binned counts by a multiplier that corresponds to the bin time used. For example, binning at intervals of 1 minute corresponds to a multiplier of 60. Also, we suggest using the Stopbar Detector sensor events in this case, since not all lanes have Advanced Detectors. We show an example of vehicle flow data produced in this manner in Figure 1.1.

Figure 3.4: Non-closed system for the SB movement at I3 Montrose Pkwy & E Jefferson Intersection, obtained from Bing Maps.

2. **Queue Length Estimation**

   Queue lengths can be estimated using a combination of Advanced Detector and Stopbar Detectors. A queue length count can be maintained for each lane, and the queue incremented each time an advanced detector is triggered, and decremented each time a stopbar detector is triggered. Since queues are with respect to vehicles approaching an intersection, Inbound Lane-Type lane sensors should be used.

   Sources of Bias. There are two sources of bias: (1) actual physical sensor bias and (2) a non-closed system. For the actual sensor bias, this happens when an advanced detector always counts more than its corresponding stopbar detector, or vice versa. Meanwhile, a non-closed system exists when there are road entrance or exits between the two detectors. A road entrance would result in a higher stopbar detection count, leading to downward bias, while a road exit would result in a lower stopbar detection count, leading to upward bias.

   We provide one case example here. See Figure 3.4, where in the SB direction, there

exists two road entrance/exits between the advanced and stopbar detectors [1]. Notice that on one side of the road, there is a commercial building, while on the other side of the road, is a housing estate. Due to the nature of these buildings at the road exits, the bias could depend on time of day as well. For example, in the morning, people who work at the commercial building will induce upward bias (as they enter the intersection, triggering an advanced detection, but do not leave by way of the stopbar detector). During the evening, when they leave work, they start inducing downward bias. On the other hand, the housing estate will show an opposite sequence of biases. In the morning, there will be downward bias, as cars leave home to go to work. Then, in the evening, as they return from work, they will enter the intersection and exit it pre-maturely, thus inducing upward bias. We show that this behaviour is present in the intersection in Figure 3.5. In the green plot, when there is no downward bias control, we see that in the morning, there is gradual downward bias, which we link to people leaving their homes. Then, at around noon, there is a sudden increase in cars. We theorize that during lunch-time, people have time off work to visit the commercial building, which happens to be the US Post Office, thus triggering the upward bias, and trickle out slowly over the next hour.

Dealing with Bias. To deal with these biases, we suggest (1) resetting the queue each time a lane/leg is detected as empty, and (2) preventing the queue from going negative. The former addresses the upward bias, while the latter deals with downward bias. To detect an empty queue, we look at the difference in time between two successive stopbar detection events. Three conditions must be fulfilled: (i) the signal phase is green or yellow; (ii) the time difference is greater than Δt, where Δt can be tuned (some possible choices are 5s, 8s or 15s) and (iii) the time difference is less than the time to the end of the phase. This is similar to the approach in [39], which detects empty queues in the same manner. In Figure 3.6, we show queues plot with

[1] This is based on the Sensor Layout Configuration by Sensys Networks, which we are not able to disclose here.

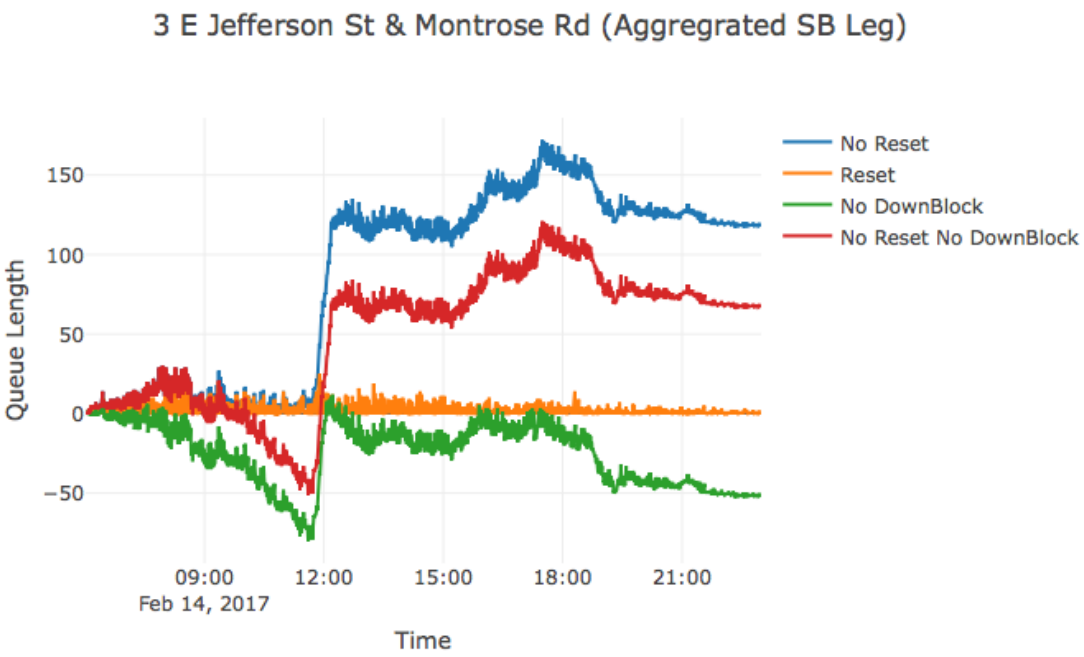


Figure 3.5: Queue Plots for Tuesday, 14 February 2017 at I3 Montrose Pkwy & E Jefferson Intersection. Blue: no upward bias control; Green: no downward bias control; Red: no bias control; Orange: both bias control in effect.

each of these bias controls in effect. What we observe is that different legs show different biases. For instance, the EB7 leg does not exhibit significant upward bias, and so the queue lengths produced from not resetting the queue and resetting the queue were very similar (Figure 3.7). On the other hand, the upward bias for NB7 leg is very apparent, necessitating a reset (Figure 3.8).

Aggregating Queues by Leg. For legs that do not have an equal number of Advanced Detectors and Stopbar Detectors, a more accurate queue length can be obtained by aggregating the queue length per leg, as opposed to per lane.

In the following Chapters, we demonstrate usage of both vehicle flow and queue length data. We show how vehicle flow data can be used for signal timing recovery and queue length data for measuring instability in the traffic network.

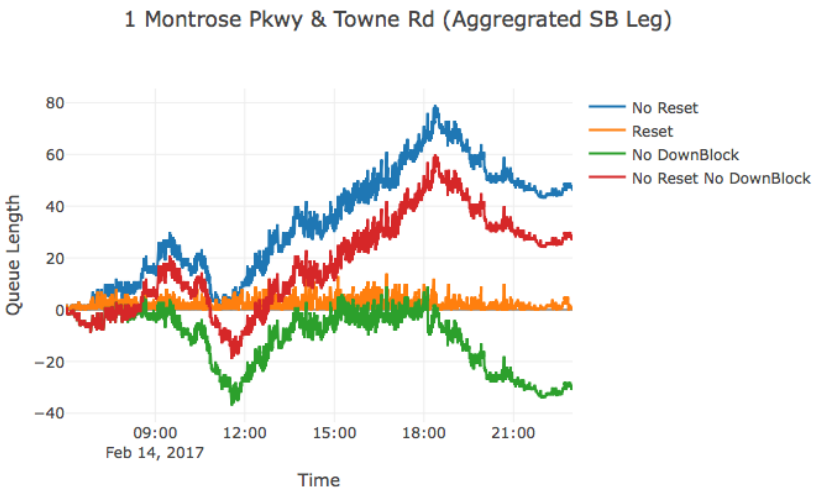


Figure 3.6: Queue lengths with different bias control in effect.

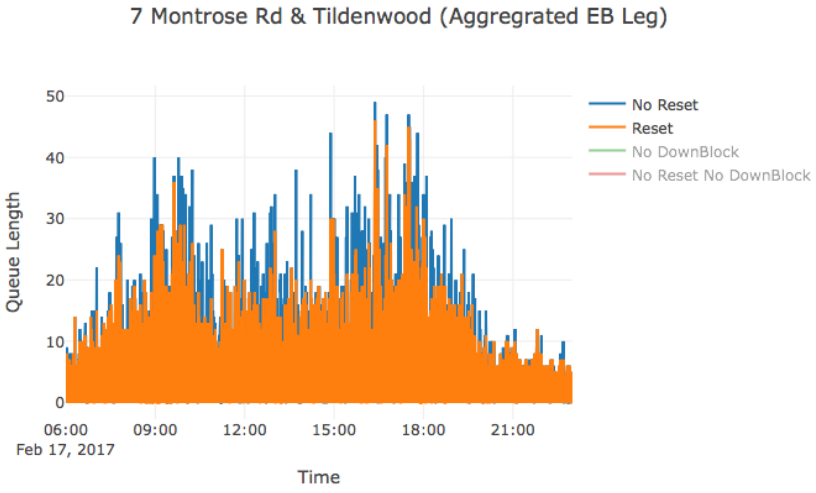


Figure 3.7: Queue lengths with low upward bias.

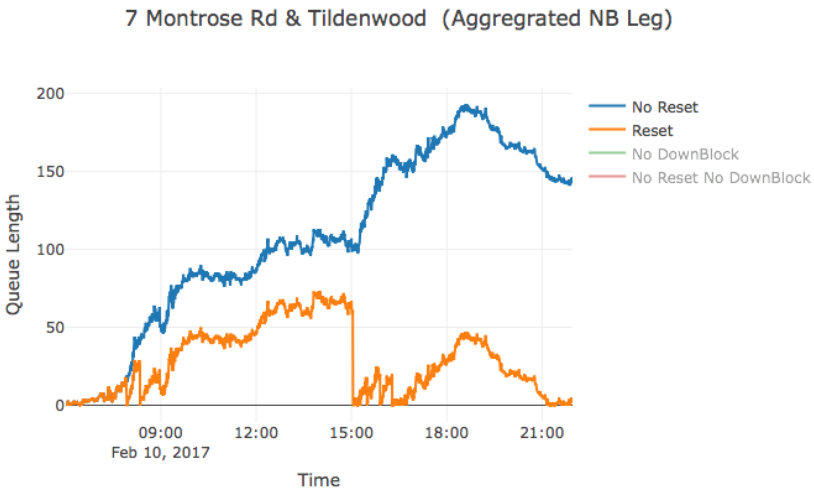


Figure 3.8: Queue lengths with strong upward bias.

CHAPTER 4
LEARNING DYNAMICS

In the previous Chapters, we built a case for the Koopman alternative framework for learning the dynamics of a system. Now in this Chapter, we demonstrate a few examples of DMD, DMD with Hankel matrices, and DMD with control in learning the dynamics of traffic. The reason we do this is two-fold. First, while DMD has shown much success in handling very high-dimensional datasets (such as the complex fluid flows in [11]), there has been relatively less treatment with regards to potentially rank deficient datasets with stochasticity [12]. Secondly (and consequently), we are interested in the performance of DMD on the traffic data we have in hand. We do this by predicting future traffic given an initial condition. In particular, we are interested in two questions: (1) can any of these algorithms learn the dynamics to the level of learning the periodicity at approximate amplitudes, and if so, (2) what is the level of accuracy of multi-step prediction? To conduct this study, we explore learning dynamics and reconstruction of the dynamics on the same day, then we test for generalization to forecast traffic on another day of similar traffic profile.

4.1 Same Day Dynamics

In this Section, we test the performance of various formulations of DMD for reconstructing future traffic states, given the same initial condition as the one used for learning A.

First, we run Standard DMD (SDMD) on queue lengths at an individual leg binned at 1 second. In this case, A is just a scalar. If the initial condition is 0, then all that propagates forward will be 0. Given that $A \leq 1$, what we observe is just a decaying trajectory, as shown in the top subplot of Figure 4.1. This is also highlighted in [12] and [25], which demonstrate that DMD on a single dimension fails to reconstruct a sinusoid.

Next, we increase the rank of the matrix by augmenting time-delay observations into

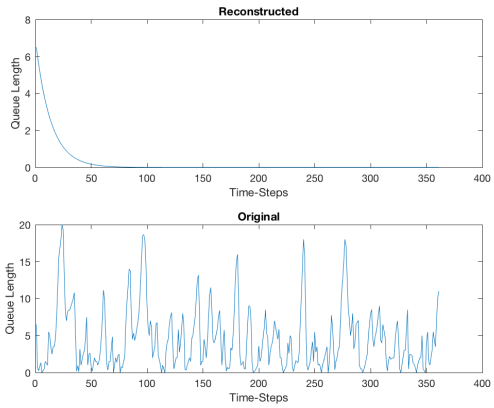


Figure 4.1: Standard DMD fails to capture dynamics.

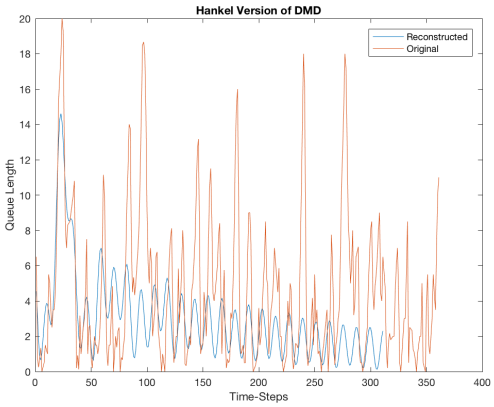


Figure 4.2: Hankel DMD shows improvement compared to Standard DMD.

the matrix, so that the data matrices takes a Hankel matrix form (HDMD). We add $h = 50$ time-delayed rows to X_1 and X_2. Then, we predict the next 350 steps. The $1s$-binned data, $h = 50$ which is 50 seconds worth of time-delayed entries, is not sufficient to capture the dynamics, and increasing h would be computationally prohibitive. Thus, we use queue lengths binned at $10s$ for this experiment. This results in an improved approximation of the dynamics, shown in Figure 4.2. However, a limitation is that the prediction does not last well beyond the number of time-appended rows added. Notice how, in Figure 4.2, the reconstruction starts to decay after the first 50 steps.

Since queue length oscillations are induced by the traffic signals, we test if the fast-decay limitation in SDMD and HDMD can be overcome by modeling signal phases as ex-

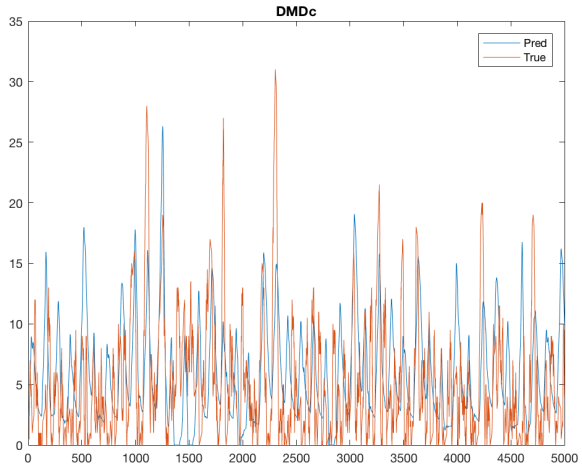


Figure 4.3: Hankel DMDc.

ogenous control input. We implement dynamic mode decomposition with control (DMDc) on queue length *and* signal phase data binned at $1s$. (To bin the signal phase events meaningfully, out of x observations that occurred within a second, we take a majority count, and forward fill the values where observations do not exist within a bin). Also, we add $h = 50$ time-delayed rows (HDMDc) appended to X_1, X_2 and $\mathbb{U}$. This method captures the dynamics the best as depicted in Figure 4.3, where we show 5000 prediction steps.

4.2 Future Day Dynamics

In this Section, we are interested if the learned A will perform well for predicting traffic on a different day. Our case study location is the WB movement at Intersection 7. We again use queue length data, binned at $1s$ for this experiment. We set up the problem as follows: learn the dynamical system using data on 14 February 2017 (Tuesday), and forecast traffic data for a similar day, 17 February 2017 (Friday). We do this for different prediction horizons T (5 minutes, 10 minutes, 30 minutes and 60 minutes), where at each prediction horizon, the algorithm is given access to the true measurement at that time. We provide a sketch of the work-flow in Algorithm 6.

Algorithm 6 Prediction

1: Learn A using data from a single day.
2: On new day:
3: **for** $k = 1$ to T **do**
4: **if** $k = 1$ (use x_1 as initial condition) **then**
5: $\hat{x}_{k+1} = Ax_k$
6: **else**
7: $\hat{x}_{k+1} = A\hat{x}_k$,
where T = forecast horizon, $\hat{x}_k$ is the predicted value at time k, and x_k is the true value at time k.
8: **end if**
9: **end for**
10: Receive observation at time T, which becomes the new initial condition. Repeat 3 until day is over.

As usual, we start with SDMD, where the state vector x_k is $x_k \in \mathbb{R}$. To additionally test if traffic conditions at other intersections improve the prediction of traffic at this leg, we also run SDMD to include legs from all intersections the network. For this case, the state vector x_k is:

$$x_k = \begin{bmatrix} x_k(1) \\ x_k(2) \\ \vdots \\ x_k(24) \end{bmatrix} \in \mathbb{R}^{24}$$

For each experiment, we form the data matrices X_1 and X_2 respectively according to Algorithm 1. We append time-delay snapshots of $h = 50$ rows for the individual leg experiment, and $h = 30$ rows for the network experiment. We use a rank truncation by removing the singular values less than a threshold of $1e^{-9}$ in the SVD of X_1.

We compare the overall performance over the day of 17 February 2017, employing the l-1 norm, averaged over the number of prediction steps, as an error metric:

$$\textit{Error} = \frac{1}{T}\sum_{k=1}^{T} |x_k - \hat{x_k}| \tag{4.1}$$

where $\hat{x}$ is the predicted measurement and x is the true measurement.

In Table 4.1, we summarize the prediction performance for the different algorithms. To interpret the error in the Table, for every prediction step, an error e of x cars is made, where

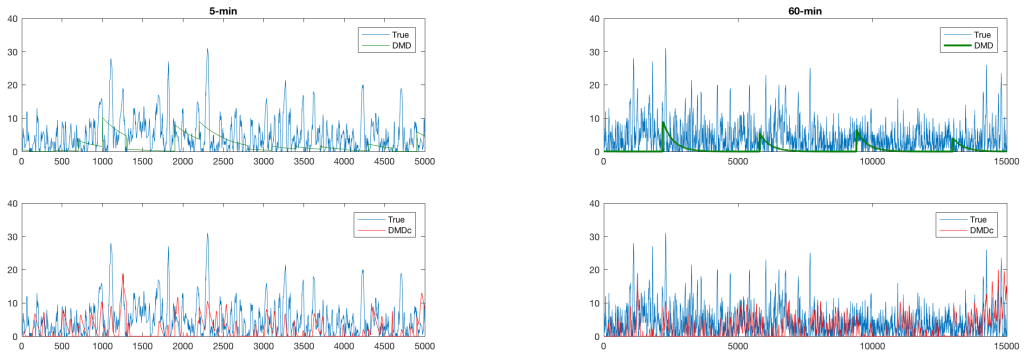


Figure 4.4: Comparing DMD and DMDc.
(Left) T = 5 min. (Right) T = 60 min.

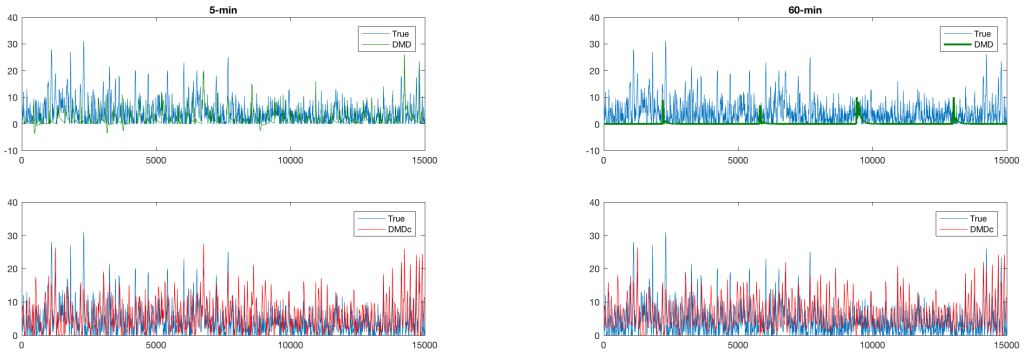


Figure 4.5: Comparing HDMD and HDMDc.
(Left) T = 5 min. (Right) T = 60 min.

e is the value given in the Table.

From the Table, we show that modeling the signal phase as an exogenous input gives a better performance boost (with the exception of when T is 5 minutes) compared to adding time-appended rows (compare SDMDc and HDMD). We reason that because the signal phases are oscillatory in nature, they provide the periodic forcing that is required to predict future states more accurately. Of all the tests, we find that HDMDc with the network performs the best.

Table 4.1: Queue Prediction Performance for New Day (l-1 norm, averaged over day) for WB7 Leg

Forecast Horizon	SDMD	SDMDc	HDMD	HDMDc	HDMD Network	HDMDc Network
5 min	4.7	4.1	3.8	3.0	3.1	**2.7**
10 min	4.7	4.3	4.5	3.3	3.1	**2.7**
30 min	5.1	4.5	5.2	3.6	3.2	**2.9**
60 min	5.3	4.6	5.5	3.7	3.2	**2.9**

4.3 Conclusion

In this Chapter, we demonstrate that adding time-delays rows is needed to learn the dynamics of the system. Though not perfect for multi-step prediction, it captures enough information about the periodicity of oscillation and trend (growth/decay) in order to analyse influence of control input during moments of instability (which will be explored in Chapter 6). Also, we remark that the effect of signal phase as control provides a significant performance boost compared to solely using time-delayed rows. This makes an interesting extension to overcoming the decaying trajectory in the standing wave example in [12].

Secondly, we test a naive algorithm for predicting traffic on a future day, given an A matrix learned for a preceding day. We show that doing so does not yield a good prediction for long horizons, but does well on shorter horizons. One reason is because the dynamics of traffic varies from day-to-day, and a further improvement that could be studied is a daily online update version of A. Nevertheless, we show that A performs well when used to reconstruct data on the same day, and on a shorter time-horizon (5-30 minutes).

In the next Chapter, we build on the Hankel formulation of DMD to understand the oscillations learned in the dynamics, and investigate the kind of information that can be obtained about the network.

CHAPTER 5
SPATIO-TEMPORAL ANALYSIS

5.1 Motivation

In this Chapter, we explore the uses of DMD to extract spatio-temporal relationships in the data to inform effectiveness of timing plans and for inference purposes. Traffic at a signalized intersection has spatio-temporal dependencies, where flow at different turn-movements are governed by the signal phasing commands. Given solely data on traffic flows, we seek to solve an inverse problem: can we infer the characteristics of the network such as cycle times, phase sequence and signal timings without having access to the timing plans? (Of course, this problem is framed under the assumptions that the signal timings caused the traffic flows we observed.) Such information might be useful for (1) companies that develop real time navigation applications to further optimize route options by coordination with signal timings and (2) predictive cruise control applications that optimize speed for fuel economy. In this case, we consider that it might be prohibitive for these companies to gain access to the actual phase timings, and thus must infer them from another way such as traffic flow data. Such an inverse problem has been studied by [40], who used low frequency bus data, and [41] who proposed a probabilistic technique to plan a vehicle's velocity given that a green light was observed at its current position.

As elaborated in Chapter 2, DMD decomposes the system into a set of dynamic modes, each growing/decaying and/or oscillating at specific frequencies as described by the eigenvalues. Eigenvalues on the unit circle are purely oscillatory modes, while eigenvalues inside and outside the unit circle are decay and growth eigenvalues, respectively. On the other hand, the modes themselves provide spatial information; the magnitude describes the amount of influence of a location in the overall mode, while the angle describes the

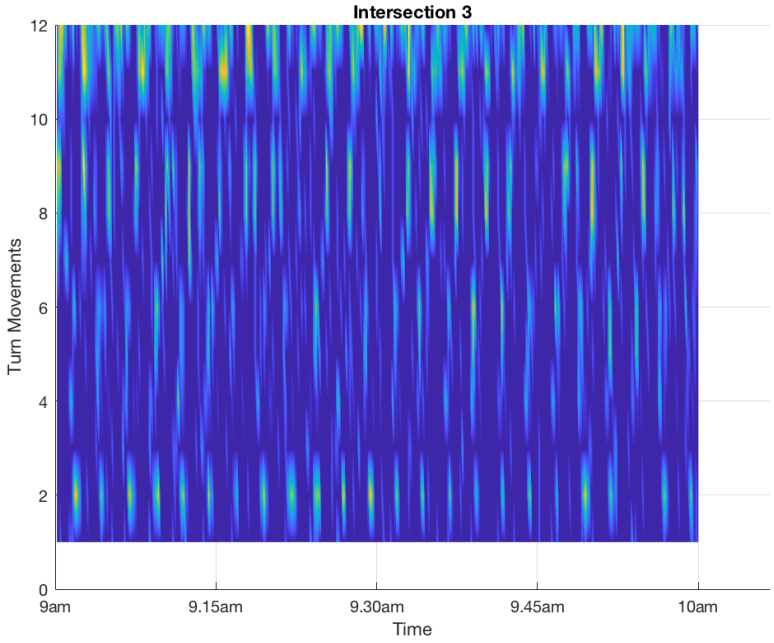


Figure 5.1: Vehicle Flow from 9-10am for Intersection 3

phase relative to other locations [17]. Applying this to traffic data, we reason that the angles could describe the relative phase sequence of the data, and turn movements in the same phase should have approximately the same angle. Additionally, we try comparing the magnitudes with percentage green time given to each movement.

[12] provides heuristics for picking modes of interest, one of which is to look at the modes with largest magnitude, i.e. based on the norm $||\psi_j||$. In some cases it is beneficial to add the effect of the eigenvalue raised to a power relevant to the number of steps taken, to account for the speed of growth/decay of the mode: $|\lambda_j|^k||\psi_j||$.

For our case, since we are interested in physically interpretable modes, i.e. the phase timings of the intersection, we study the modes corresponding to the frequencies of interest.

5.2 Cycle Times

As a litmus test for DMD, we are interested in the frequencies extracted in the learned A matrix. At the very least, it should contain the frequencies corresponding to the cycle time

Table 5.1: First r = 10 eigenvalues when running DMD on vehicle flow on two different cycle times on 14 February 2017 at Intersection 3

Time: 9-10am		**Time: 10am-3pm**	
λ	Period (s)	λ	Period (s)
1	53	1	43
2	-53	2	-43
3	74	3	-
4	-74	4	120
5	-	5	-120
6	-	6	-
7	150	7	60
8	-150	8	60
9	116	9	75
10	-116	10	75

of the intersection, where a cycle time is defined as the time for the intersection to complete all of its phases. As ground truth, we have the timing plans of the intersection in question.

Our study day is 14 February 2017. Based on the signal tables for one of the intersections, there are two different cycles on this day; 120s (10am-3pm and 7pm-midnight) and 150s (6am-10am and 3pm-7pm). We run DMD on a Hankel matrix of vehicle flows, binned at 10s, on non-overlapping cycles.

First, we run a rank-truncated DMD, with $r = 10$, on an hour's worth of flows between 9am-10am at Intersection 3, which corresponds to a 150s cycle. Since there are 12 turn-movements, $x \in \mathcal{R}^{12}$. An hour's worth of data at 10s bins corresponds to N = 360 samples. Figure 5.1 shows a plot of vehicle flows during this hour. Then, we run the same experiment, but on a longer set of flows, from 10am-3pm, which corresponds to the 120s cycle. In this case, N = 1800 samples.

Table 5.2: Timing Plan for Intersection 3, as obtained from MCDOT. (Left) 6am-10am, 150s cycle. (Right) 10am-3pm, 120s cycle. Phases 1, 2, 5 and 6 are in Barrier 1, while Phases 3, 4, 7 and 8 are in Barrier 2.

Phase	1	2	3	4	Phase	1	2	3	4
Time (s)	26	53	21	50	Time (s)	16	30	35	39
Phase	5	6	7	8	Phase	5	6	7	8
Time (s)	26	53	21	50	Time (s)	16	30	35	39

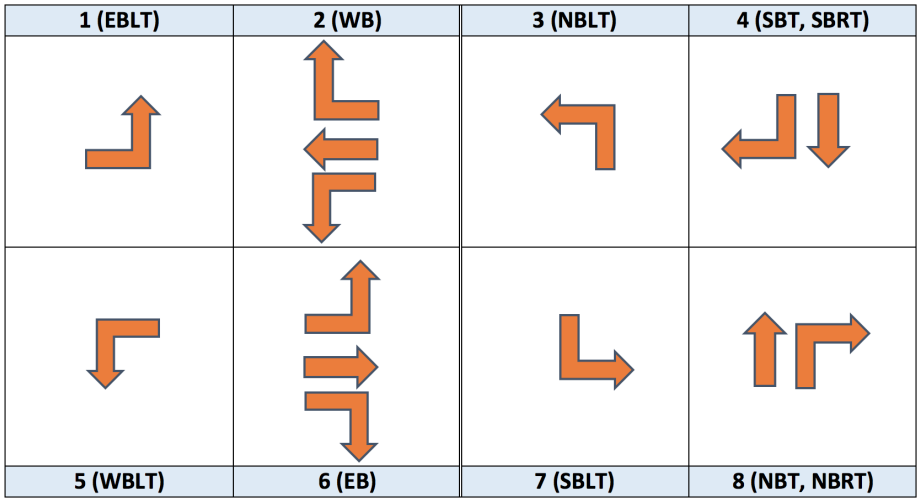


Figure 5.2: Ring and Barrier Diagram for Intersection 3

Table 5.1 shows the results from these two experiments, where we show the first 10 eigenvalues with the imaginary part converted to seconds, sorted according to magnitude. The conversion formula we use is:

$$t = \frac{2\pi}{\Im\{\lambda_j\}}\Delta t \tag{5.1}$$

where $\Delta t = 10s$.

From Table 5.1, the most interpretable periods are the ones corresponding to the phase cycle for this intersection at this time. (Refer to Table 5.2 for the true timing plans and Figure 5.2 for the ring and barrier diagram denoting the movements for each phase). The other periods are not as clear, however, but we provide some reasoning. The intermediate periods could correspond to the total time for a particular barrier [1], where the 74s in the 9-10am cycle could correspond to either barrier, while in the 10am-3pm cycle, the 75s corresponds to the NB-SB barrier, and the 43s corresponds to the EB-WB barrier. On the other hand, smaller periods may correspond to the timings for different phases, for example, in the 9-10am cycle, the 53s period corresponds directly with the WB movements. The 60s

[1] A barrier is used to separate conflicting movements, i.e East-West movements from North-South movements [42]

Table 5.3: First r = 10 eigenvalues when running Hankel DMD on vehicle flow from 9am-10am on 14 February 2017 at SBT3 movement

λ	Period (s)
1	38
2	-38
3	50
4	-50
5	75
6	-75
7	150
8	-150
9	-
10	-

in the 10am-3pm cycle may represent the start-stop nature of cars clearing the lanes upon a green light. We also observe that there is no direct correspondence an individual turn-movement for the 10am-3pm experiment, suggesting that a shorter number of samples is needed to extract the finer periods.

Next, we run DMD on an individual turn-movement to test if phase times can be extracted in addition to the cycle times. We do this on flows at the SBT3 movement from 9-10am. In Table 5.3, we see that the 50s phase cycle is present, as well as the 150s intersection cycle time. This can be cross-checked with the true phase time for Phase 4 in Table 5.2.

For comparison, we also run a Fast Fourier Transform (FFT) on the data. Figure 5.3 shows the frequencies obtained from the FFT on the vehicle flows at the SBT movement at Intersection 3 from 9-10am. The three dominant frequencies are 150s (cycle time), 75s (SB-NB barrier) and 50s (SBT green time).

We conclude that DMD is able to recover the natural frequencies that exist in the system, as the frequencies are comparable with those produced by an FFT, and correspond to the true cycle times encoded in the intersection. Of course, if we just wanted to recover the frequencies of the system, one could argue that using an FFT would achieve the same purpose. However, DMD additionally provides spatial context for each frequency, as we

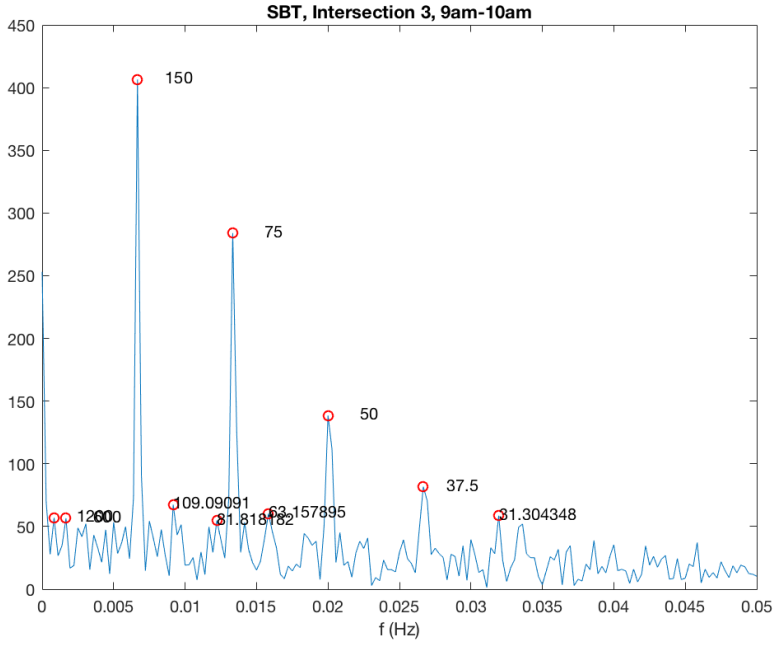


Figure 5.3: FFT on SBT3 movement, for Intersection 3.

will see in the following section.

5.3 Barrier and Phase Recovery

In this section, we further investigate the phase angles of the modes. We are interested if the mode ψ_j corresponding to the eigenvalue λ_j at phase cycle frequency would reveal the phase sequence in the system. This idea flows naturally from the fact that vehicle flows at an intersection are oscillatory as induced by the phase timings encoded in the system. We set up the DMD experiment in the same manner as the previous section, using a Hankel data matrix of vehicle flows from 14 February 2017, and then study the angles of the mode corresponding to the 120s cycle.

To convert the angles to seconds, we used the following formula:

$$t = \frac{\angle\{\psi_j\}}{2\pi}\delta t \tag{5.2}$$

where δt is the corresponding cycle time, for example, $120s$.

We provide a visualization of phase angles in colour overlaid over a Google Maps [2] representation of the intersection. In the diagram, lines in the center correspond to Through movements, and the accompanying two lines represent the Left-Turn and Right-Turn movements, depending on their location. For example, in the East-Bound leg, the first line represents the Left-Turn movement, the middle line represents the Through Movement, and the third line represents the Right-Turn movement. We produce the colour visualizations using the positive shifted phase, where we shift each angle by the maximum value, as illustrated by the bottom plot in Figure 5.4, so that the values wrap around by 120. In this Figure, we can see that the relative difference between the phases correspond to the phase timings for the individual movements. For example, the difference between the SBT and WBT movement is about 50s, which is comparable to the time elapsed between Phases 4 and 2 shown in Table 5.2. On the other hand, we can see that cars arrive at different times for the SBT and NBT movements. From 5.2, we see that the green times for the SB and NB movements occur simultaneously. However, in Figure 5.4, the NBT and SBT differ by about 20s.

From experiments, when a longer window (6 hours) of samples is used to build the data matrix, or, when vehicle flows are not consistently high, the SB-NB and EB-WB barrier is extracted. In Figure 5.7, notice how EB-WB movements are clearly grouped in blue, while most of the SB-NB movements are grouped in red.

In order to recover the individual phase timings, a shorter window (15-30 minutes) needs to be used, and sufficiently consistent flow of traffic. We re-run DMD on a 30 minute window in the morning peak hour (9.30-10.00am). In Figure 5.6, we can observe non-conflicting phases corresponding with each other. For example, the NBLT and SBLT movements clearly correspond to each other, while the NBT, NBRT correspond with the SBT movement. These results show coherence to the actual ring and barrier diagram of this intersection in Figure 5.2.

Next, we examine the magnitude of the modes. We compare them with the percentage

[2] To interface MATLAB with the Google Maps API, we make use of code from [43].

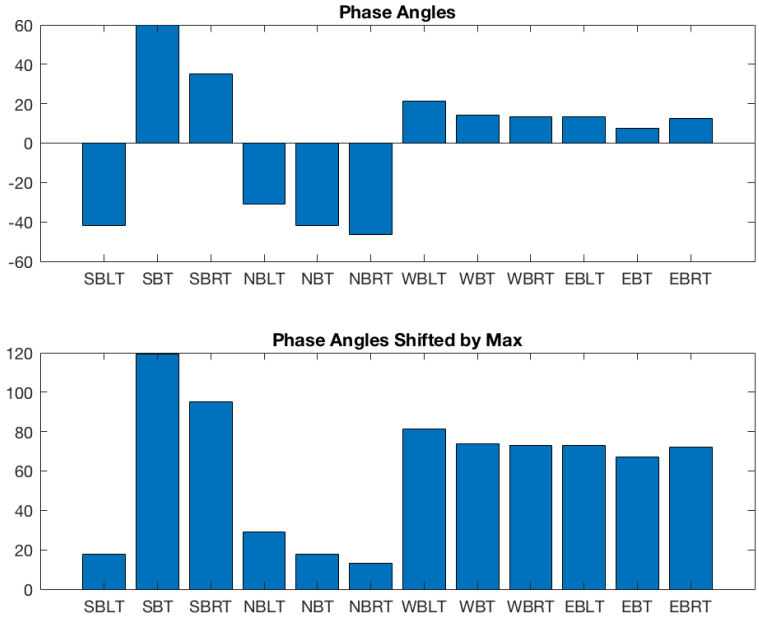


Figure 5.4: Original phase angles (s) and shifted phase angles (s) for Intersection 3, for 120s cycle and 6 hours worth of samples.

green times for each movement, and suggest that this provides some insight to whether the amount of green time is appropriately assigned. A higher magnitude might indicate need for a longer green split, since a higher value could mean a greater utilization of its green time. Refer to Figure 5.8, where we show the mode corresponding to the 120s eigenvalue.

5.4 Conclusion

In this Chapter, we have shown how DMD can be used for extracting the cycle times of an intersection and recovering the relative phase sequences of each movement. We must point out that one limitation of this method is that it requires consistent vehicle flow from all turn-movements at the intersection during their green phases.

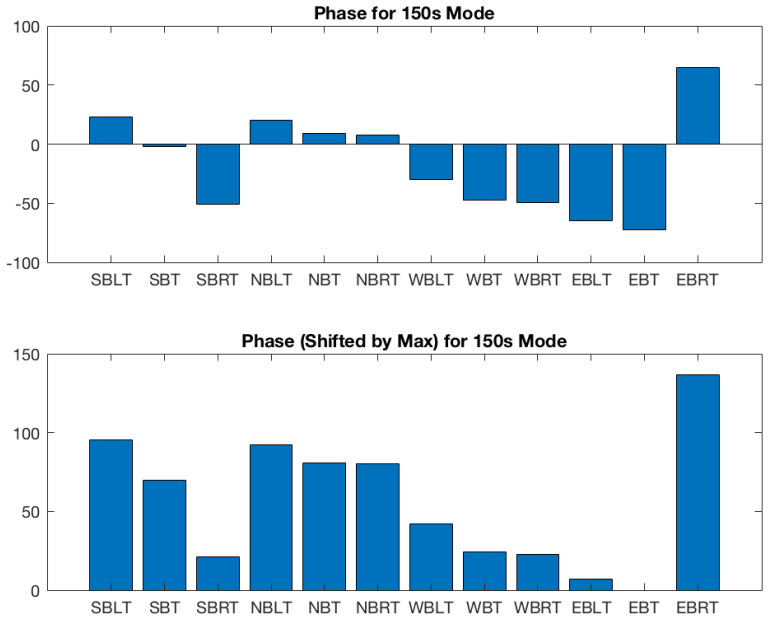


Figure 5.5: Original phase angles (s) and shifted phase angles (s) for Intersection 3, for 150s cycle and 30 minutes worth of samples.

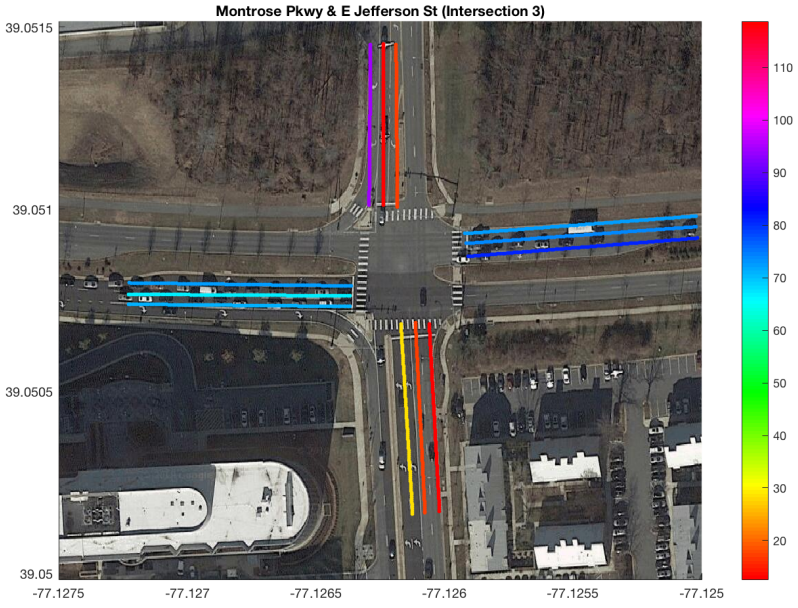


Figure 5.6: Phase angles corresponding to 120s mode, when running DMD using 6 hours worth of samples. Notice how the colours display the barrier of the phase control.

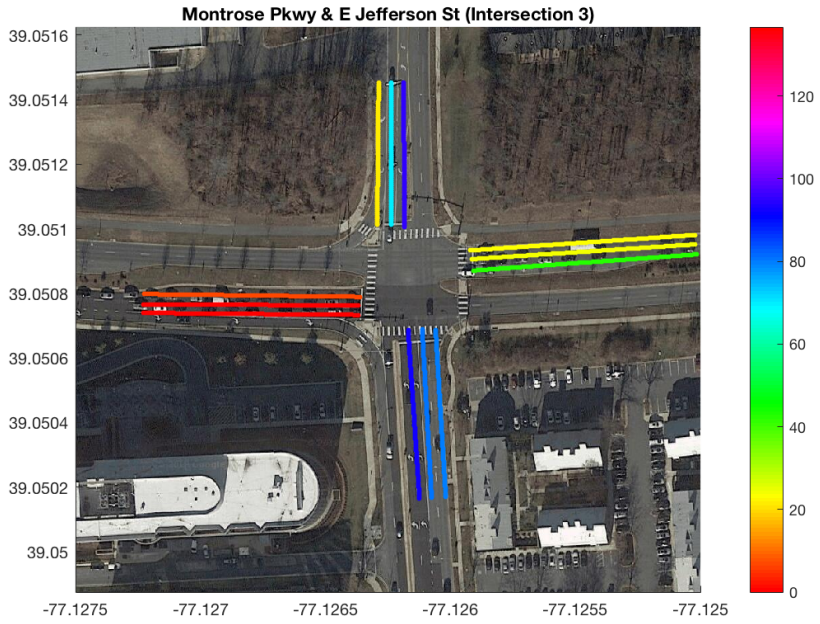


Figure 5.7: Phase angles corresponding to 150s mode, when running DMD using 30 minutes worth of samples during the morning peak hour.

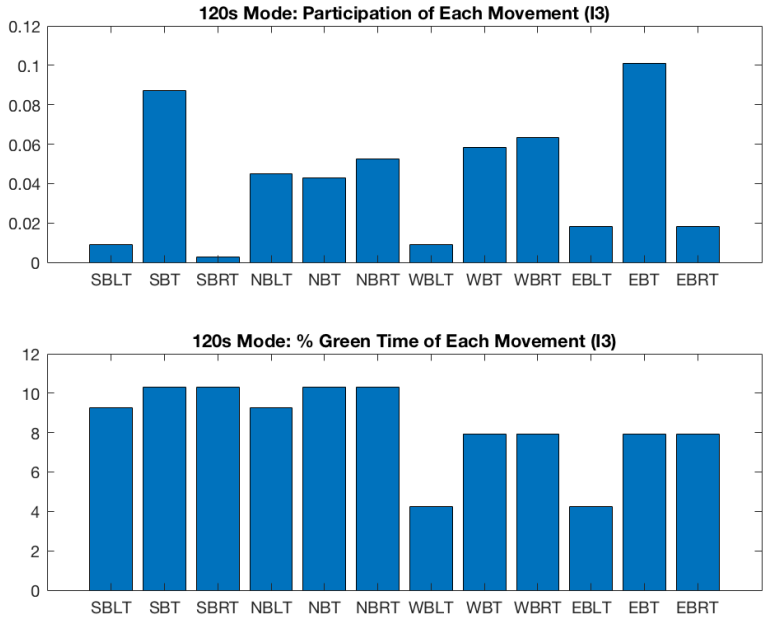


Figure 5.8: A comparison of the magnitude of each movement in the 120s mode and the percentage green time.

CHAPTER 6
INSTABILITY AND CONTROL

6.1 Motivation

Queue lengths at traffic intersections exhibit oscillations according to cycle times, where they build up during a red phase and clear out during a green phase. During peak hours, queues may accumulate if the clearance interval is not sufficient to completely discharge the queue. This condition is known as cycle failure [44]. Traffic accidents and lane closures are also causes of cycle failure.

When a queue breaks down at an isolated leg at an intersection, it would be advantageous to have a responsive system that serves a more effective signal phasing to clear the queue faster. In traditional systems, signal phase timings are not updated immediately. The process is normally initiated with a citizen complaint, which triggers a series of field observations, data collection and software simulation before changes are made [45]. This procedure, though effective for long-term timing strategies, can be made more flexible to include short-term changes during abnormal events.

Moreover, when a new timing plan is proposed, a software simulator, such as Synchro, is used for verification. The simulation is based on user settings of arrival and departure rates that is normally modeled according to a Poisson process. This approach however, is dependent on how good of an approximation the model is to the true intersection. Furthermore, abnormal behaviour is difficult to model, being rare events. Once again, while using a software package is a good approach for long-term planning, developing a simulation strategy that is based on actual data would improve prediction performance.

With this in mind, we explore in this Chapter, the detection of unstable queue dynamics, primarily, for identifying traffic accidents. Secondly, given a queue breakdown, we

investigate using exogenous control input to effect change on the queue dynamics.

6.2 Instability Detection Algorithm

In the opening of this Chapter, we argued that changes should be made to the network system to be able to respond in real-time to abnormalities. However, one must be able to detect that such an abnormality has occurred in the first place. In this section, we propose an algorithm for identifying unstable dynamics in queue lengths. Here, we clarify the intended usage of the algorithm. We want to detect abnormally sustained queue growth rates that correspond to an abnormal event, such as a traffic accident, and not the regular peak hours. We propose to do this by finding unstable eigenvalues, i.e when A contains $|\lambda| > 1$. However, we also expect queue growth from the natural build-discharge corresponding to the phase cycle, as well as from cycle failure during peak hours. The A matrix would also contain unstable eigenvalues in those cases. It is important that the algorithm is able to differentiate between a traffic accident and these two normal events. We find that an important discriminating characteristic is not only the presence of an unstable eigenvalue, but also the number of consecutive unstable eigenvalues encountered. We summarize this in Algorithm 7.

Algorithm 7 Instability Detection Algorithm

1: Initialize ctr = 0.
2: Choose $\tilde{n}$, the number of samples to look back.
3: Start at time $k = \tilde{n}$.
4: Run DMD to learn A using observations from k-$\tilde{n}$ to k.
5: Find the largest $|\lambda|$ of A and store it. If $|\lambda| > 1$, increment ctr. Else, set ctr = 0.
6: Go to the next sample. Repeat steps 4-5 until there are no more samples. If $ctr > \epsilon$, then flag the incident as an abnormal growth, where ϵ is a threshold value.

6.3 Instability Experiment

To compute A, we run DMD sequentially on the previous $\tilde{n}$ samples of queue lengths at the leg of interest. (To learn about how we obtained the leg-aggregated queue lengths, refer to Chapter 3).

A few important parameters that need to be chosen are:

1. The number of samples, $\tilde{n}$.

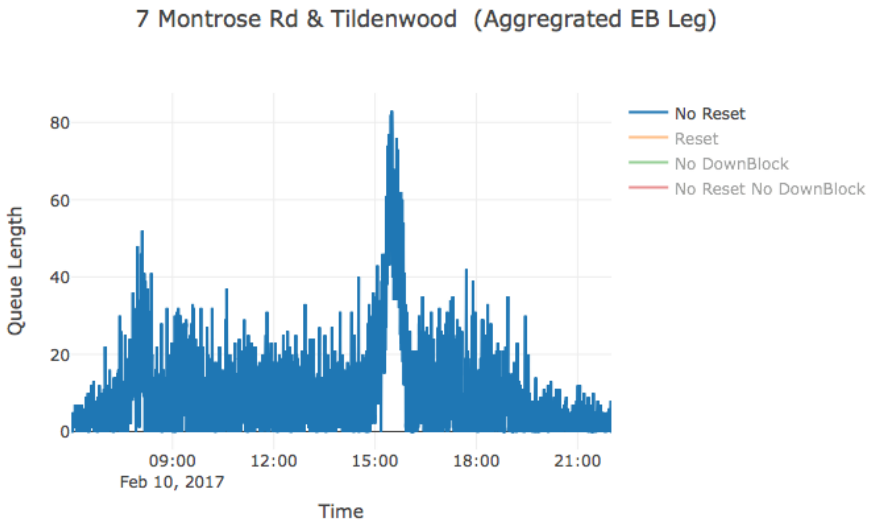
   The number of samples is related to how long of a time-frame to model the dynamics. Since accidents may have sustained growth compared to peak hour growth, it might be more reasonable to have a longer $\tilde{n}$ such as 30 minutes.

2. The number of time-delayed rows to add in the data matrix, h.

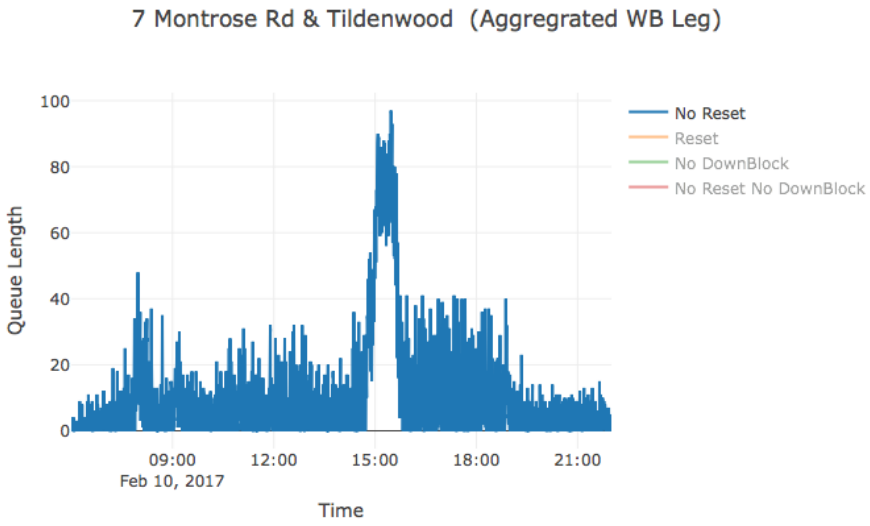
3. The rank truncation in the DMD algorithm, r.

Our investigation day is 10 Feburary 2017. Figure 6.1 shows the queue lengths on that day, where there was a reported accident at 2.47pm, at the Tildenwood-Montrose Road Intersection (I7) which affected the WB and EB movements. We obtain the crash data from the Maryland Open Data Portal [46]. Notice the prolonged spike from the time of the accident till about 4pm, which is more severe compared to the normal morning peak hour. For comparison, we also plot queue lengths for a normal Friday, 17 February 2017 in Figure 6.2.

We run DMD on aggregated queue length binned at 1s, with $h = 10$ and $r = 10$. Using Algorithm 7, we compare the number of unstable eigenvalues for a normal day and an accident day. We show that the number of consecutive unstable eigenvalues is at least 5 times larger during an accident (Figure 6.4), compared to normal peak hours (Figure 6.3).

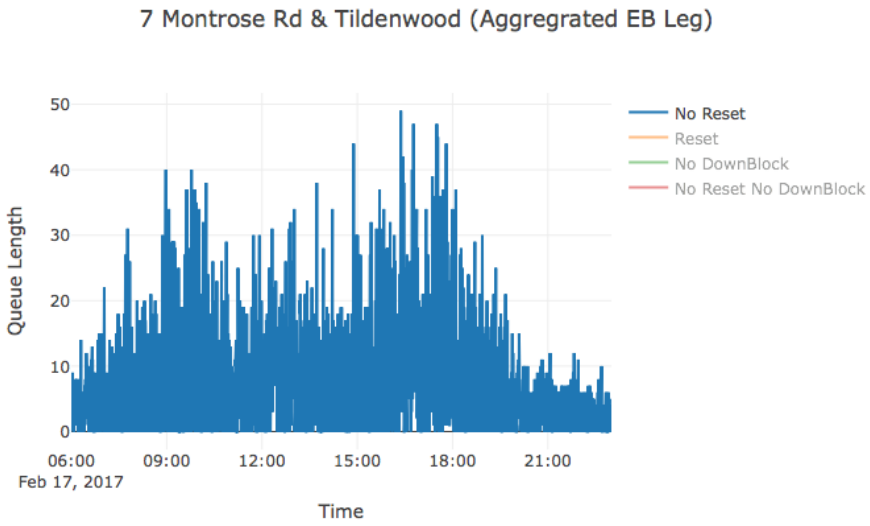


(a) Queues for EB7

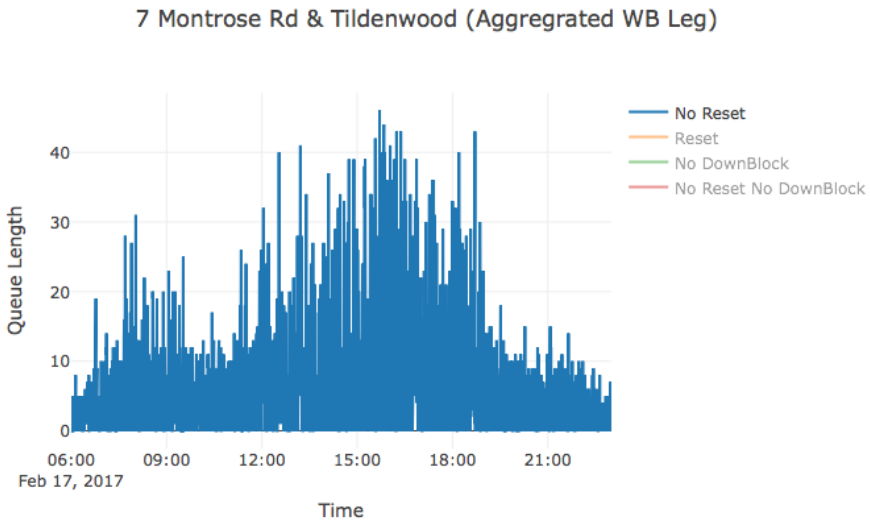


(b) Queues for WB7

Figure 6.1: Queue Lengths on 10th February 2017 at Intersection 7, where there was an accident at 2.47pm. Notice the increased spike in queue lengths during the time of accident.

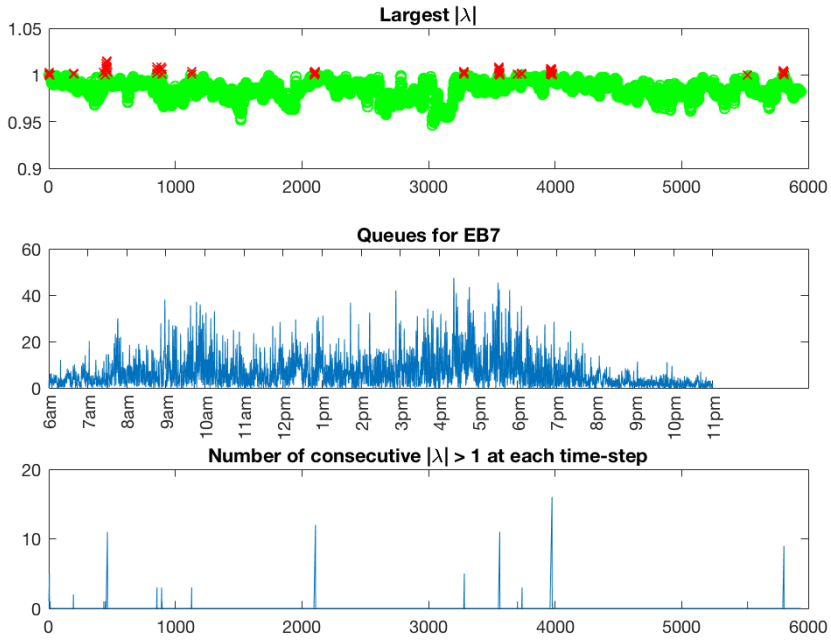


(a) Queues for EB7

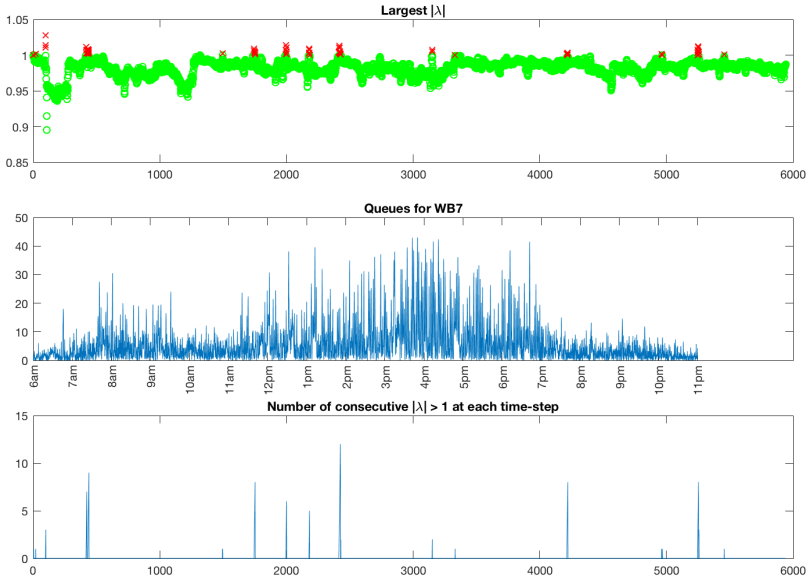


(b) Queues for WB7

Figure 6.2: Queue Lengths on 17th February 2017, a normal day.

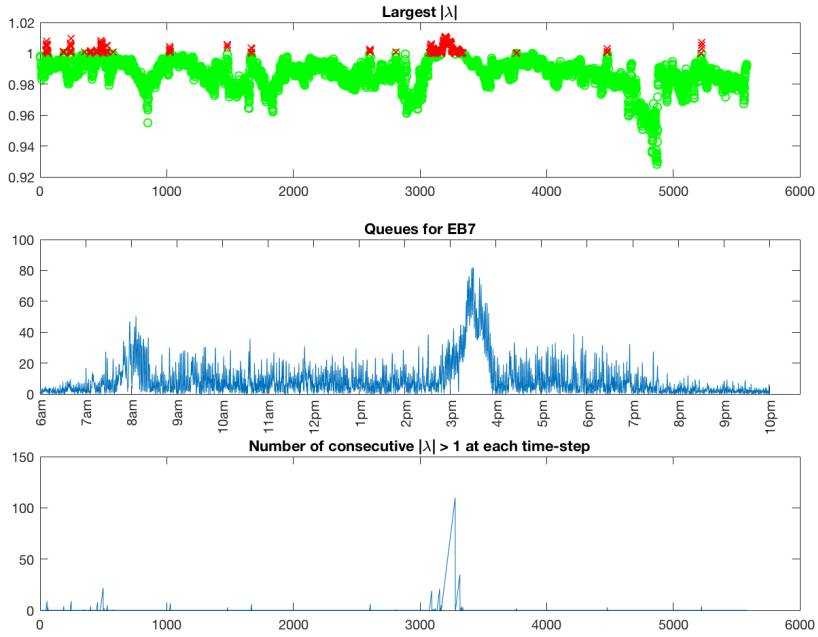


(a) Unstable eigenvalues for EB7

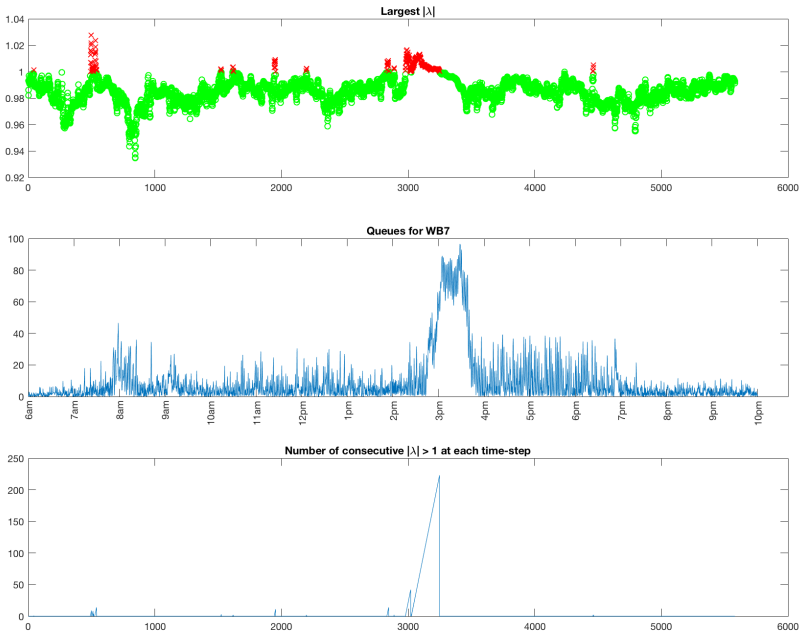


(b) Unstable eigenvalues for WB7

Figure 6.3: 17 February 2017 (normal day). (Top) The largest $|\lambda|$ obtained at each time-step (unstable λ are shown in red, while stable λ are blue.) (Middle) Queue plot. (Bottom) Count of consecutive unstable λ.



(a) Unstable eigenvalues for EB7



(b) Unstable eigenvalues for WB7

Figure 6.4: 10 February 2017 (accident at 2.47pm). (Top) The largest $|\lambda|$ obtained at each time-step (unstable λ are shown in red, while stable λ are blue.) (Middle) Queue plot. (Bottom) Count of consecutive unstable λ. Notice the number of consecutive unstable eigenvalues is much larger during the accident.

6.4 Instability Mitigation

Control becomes a motivating factor during moments of instability. Given a growing queue length at an isolated leg approaching unstable conditions, is it possible to inject control input to mitigate the instability by adjusting the signal phases?

We set up the problem as follows. Define the input vector u_k at time k as the signal phase at each turn movement at an intersection:

$$u_k = \begin{bmatrix} u_k(1) \\ u_k(2) \\ \vdots \\ u_k(12) \end{bmatrix}$$

where

$$u_k(i) = \begin{cases} 0, & \text{if } u_k(i) = \text{red} \\ 1, & \text{if } u_k(i) = \text{green or yellow} \end{cases}$$

The state vector x_k at time k consists of aggregated queue lengths at the affected leg at the intersection: $x_k \in \mathbb{R}$.

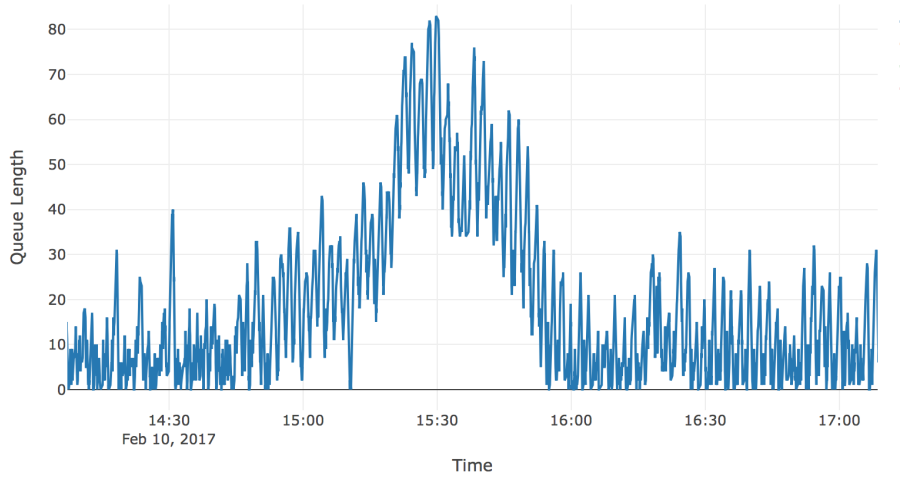
Then, form the data matrices X_1, X_2 and $\mathbb{U}$ respectively according to Algorithm 2. We append time-delay snapshots of $h = 12$ rows, such that X_1, X_2 and $\mathbb{U}$ are Hankel matrices. Additionally, we use a rank truncation based on the number of singular values above $1e^{-10}$ in the SVD of both Ω and X_1.

Notice that in Figure 6.5, the queues formed from the accident start to clear at 3.30pm. Interestingly, we note that the signal phase changes from the 110s cycle to 120s cycle at 3.30pm (see Table 6.1 for the timing plans). What is additionally interesting is that in the 120s cycle, *the green times of the WB-EB phases are extended*. We are curious if the signal phase had any effect in clearing the queue, and not due to fewer vehicles entering the queue at 3.30pm. We utilize DMDc to carry out this investigation.

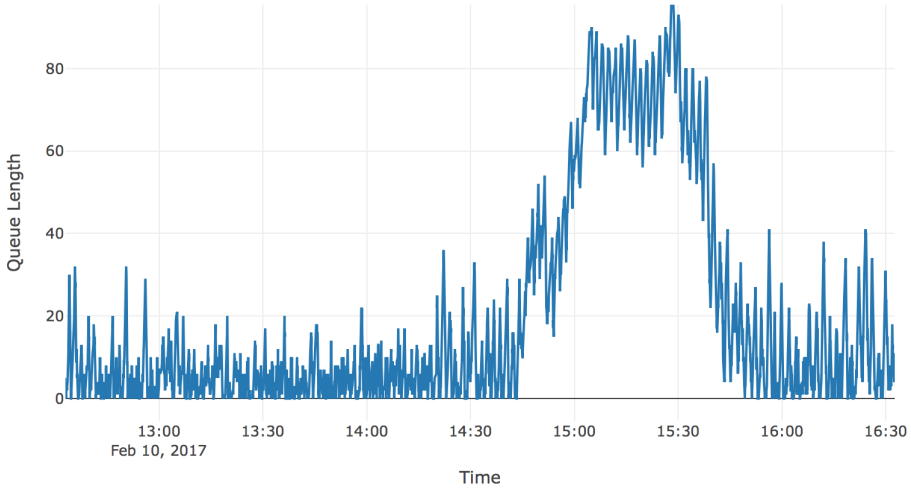
Table 6.1: Timing Plan for Intersection 7. (1-EBLT, 2-EB, 4-SB, 5-WBLT, 6-WB, 8-NB). (Left) 9.30am-3.30pm, 110s. (Right) 3.30-7pm, 120s. Phases 1, 2, 5 and 6 are in Barrier 1, while Phases 4 and 8 are in Barrier 2.

Phase	1	2	4	Phase	1	2	4
Time (s)	16	49	45	Time (s)	23	53	44
Phase	5	6	8	Phase	5	6	8
Time (s)	16	49	45	Time (s)	23	53	44

First, we learn the A and B matrices using the queue lengths (i) from 2.30pm to 3.59pm (starting time before the accident occurred) and (ii) from 2.50pm to 3.59pm (starting time after the accident occurred), with the *original* signal phase as input. We predict future queue lengths, using x_1 to be the queue lengths at 2.30pm and 2.50pm respectively, and compare the predicted values with the true values for validation (see the top subplots of each Figure in Figure 6.6). Then, we change the signal phase to the 3.30pm cycle as the new input *from the beginning*, and project queue lengths using the same initial conditions. Notice how, with the effect of the modified signal phase, the queue is mitigated faster, before 3.30pm (see bottom subplots of each Figure in Figure 6.6). This has profound implications: we have shown that using DMDc, the effect of extended green times for the relevant leg during an accident can result in faster clearing of the queue.

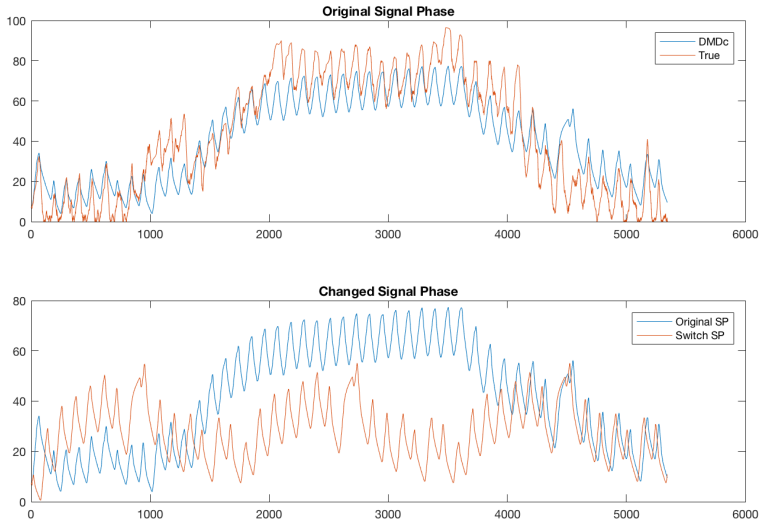


(a) EB7

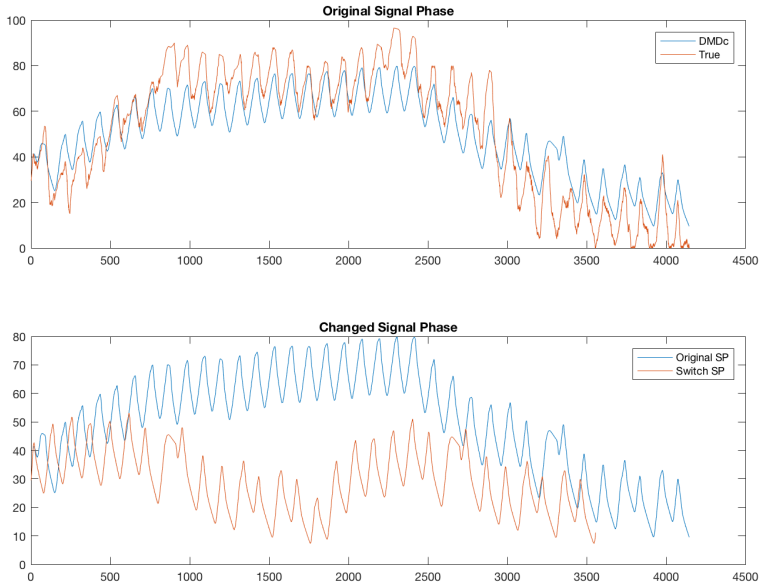


(b) WB7

Figure 6.5: Zoomed-in Queue Lengths during the accident. Notice that the queue starts to clear at 3.30pm.



(a) 2.30-3.59pm



(b) 2.50-3.59pm

Figure 6.6: Using DMDc to simulate effect of new signal phase to mitigate queue on the WB7 leg on different time-ranges.

CHAPTER 7
CONCLUSION

7.1 Summary of Findings

This thesis was the result of approaching the problem of traffic analysis and control through the lens of Koopman Operator Theory. We focused more on the linear operator data-driven algorithm of this theory, namely, dynamic mode decomposition. We showed that one strength of dynamic mode decomposition is in extracting distinct oscillatory modes. Using this, we were able to solve the inverse problem of traffic timing recovery using vehicle flow data.

Also, while dynamic mode decomposition as presented in its original form was inadequate for completely capturing the dynamics of our system, especially with data of high resolution ($\leq 9s$), we showed how techniques such as adding time-shifted observations and adding a control input improved the model substantially. With a good model in hand, we were able to implement an algorithm for real-time queue breakdown detection. Following that, we demonstrated a method for real-time signal phase updating based on the detected instabilities.

7.2 Future Work

7.2.1 Operator and Spectral Theory Analysis

In Chapter 2, we described how DMD and Arnoldi solve the rank-deficiency problem in different ways; by using the larger C matrix, appending time-shifted values (Hankel matrix, Vector Prony), and taking nonlinear mappings of observations (Extended DMD). These variations can be viewed as different finite-dimensional subspaces on which to project the modes of the Koopman Operator. Further work could done to conduct a thorough mathe-

matical analysis of how each of these variations result in different approximations.

7.2.2 Online Learning

In Chapter 4, we compared several variants of DMD in learning traffic dynamics, and provided a simple example for using the learned A on a different day. It would be an interesting extension to develop an online learning algorithm with DMD, and conduct a regret [47] analysis.

7.2.3 Establishing Theory for Instability Mitigation

In Chapter 6, we showed that by using a set of signal phases with extended green times, we were able to mitigate unstable queue lengths in a leg affected by an accident. Further analysis could be done to establish (1) instability classification; the severity of the queue growth at the affected leg, (2) extent of instability; the number of legs affected, and (3) the appropriate signal phase to be synthesized for each case. This can be summarized in the following question: "Given an instability condition (queue spill-back), does there exist a set of signal phase timings as control input to alleviate the queue?"

Appendices

APPENDIX A

DMD LIBRARY AND GRAPHICAL USER INTERFACE IN PYTHON

We have constructed a library for DMD in Python, which contains classes for DMD, Hankel version of DMD, Companion DMD, Vector Prony, and Arnoldi. In tandem with this, we have made a graphical user interface (GUI), to facilitate visualization of DMD modes with different algorithms, number of samples and turn-count data. Subject to data disclosure approval, we may make the GUI available on Github. We provide screen-shots of the GUI below in Figures.A.1 and A.2.

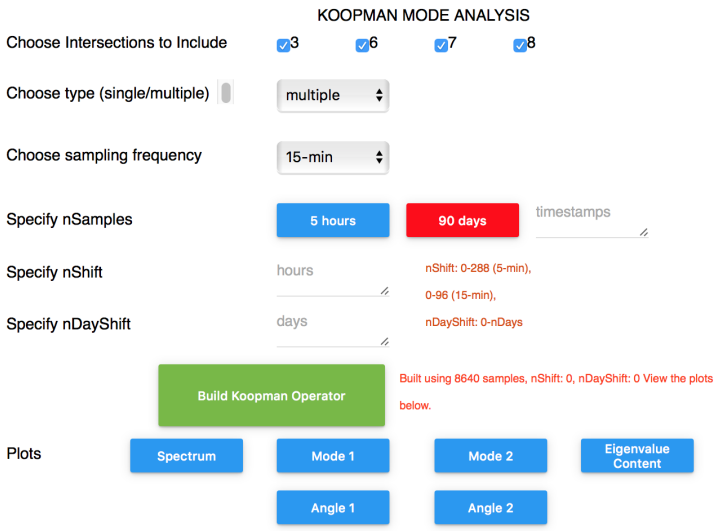


Figure A.1: GUI Control Panel

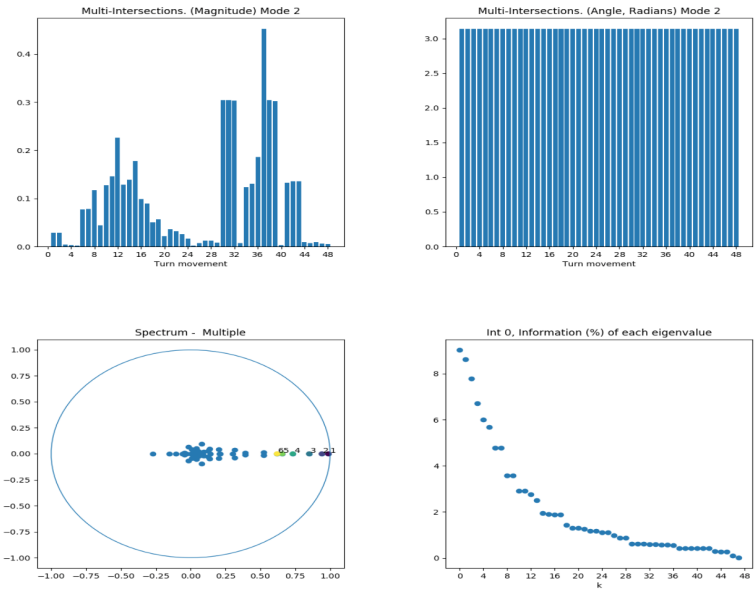


Figure A.2: GUI Visualization Panel

APPENDIX B

DATABASE DOCUMENTATION

We have produced an SQLite database of the two year's worth of traffic data, of size 250 GB. We provide the database schema and mapping tables here, along with some example query code in Python. Further improvements include: (1) adding other data such as turn-count and travel time data, which has been pulled from the Acyclica API, and (2) sorting the data according to time-stamps to improve searching efficiency.

Database Schema

Column	1	2	3	4	5
Field	intersection	intersection_name	num_inbound	num_outbound	numTM
Datatype	Integer	Text	Integer	Integer	Integer
Note	Primary Key	-	-	-	-

Table B.1: Network Table

Column	1	2	3	4	5	6
Field	intersection	legID	legDir	numInb	numOutb	speedLimit
Datatype	Integer	Integer	Text	Integer	Integer	Integer
Note			SB,EB, WB,NB	# inbound lanes	#outbound lanes	mph -

Table B.2: Leg Table

Column	1	2	3	4	5	6
Field	intersection	legID	laneID	isPocket	length	type
Datatype	Integer	Integer	Integer	Text	Integer	Text
Note			for within leg (number)	true or FALSE		inbound or outbound

Table B.3: Lane Table

Column	1	2	3	4	5
Field	intersection	legID	laneID	laneDir	laneMov
Datatype	Integer	Integer	Integer	Text	Text
Note				SB,EB,WB,NB	LT,RT,T

Table B.4: Mov (Movement) Table

Column	1	2	3	4	5	6
Field	intersection	barrier	ring	phaseNum	phaseMov	info
Datatype	Integer	Integer	Integer	Integer	Text	Integer
Note					EBLT etc	0 = protected 1 = permissive

Table B.5: Signal Phase Table (code written, not yet updated .db file)

Column	Field	Datatype	Explanation
1	intersection	Foreign Key	-
2	event_type	Text	SP = signal phase event
3	sp_validity	Integer	Signal Phase Validity (1 = valid, 0 = not)
4	tstart_epoch	Numeric	Time of event start in Epoch (UTC)
5	tstart_local	Text	Col 4 in time local to position of sensor
6	seq_num	Numeric	Sequence Number (unsigned long)
7	cycle_num	Integer	Estimated cycle number
8	barrier_num	Integer	Currently active barrier
9	num_rings	Integer	Number of rings
10	ph1_active	Integer	Phase number of the active phase in ring
11	ph2_active	Integer	-
12	ph1_state	Integer	Active Phase State of phase i
13	ph2_state	Integer	1 = yellow
14	ph3_state	Integer	2 = green
15	ph4_state	Integer	0 = red
16	ph5_state	Integer	-
17	ph6_state	Integer	-
18	ph7_state	Integer	-
19	ph8_state	Integer	-
20	ph9_state	Integer	-
21	ph10_state	Integer	-
22	ph11_state	Integer	-
23	ph12_state	Integer	-
24	ph13_state	Integer	-
25	ph14_state	Integer	-
26	ph15_state	Integer	-
27	ph16_state	Integer	-

Table B.6: Signal Phase Events Table

Column	Field	Datatype	Explanation
1	intersection	Integer	Foreign key
2	event_type	Text	Lane Events, Stopbar = LCS Lane Events, Advanced = LCA
3	legnum	Integer	Leg number
4	lanenum	Integer	Lane number
5	lanetype	Integer	1- inbound, 2 - outbound
6	t_last_epoch	Numeric	Time of last event (last undetect)
7	t_last_local	Text	Col 6 in local time
8	ph_active	Integer	Binary coded phases active
9	sp_valid	Integer	Signal phase valid: 1 = valid, 0 not
10	sp	Integer	Signal phase: 1-yellow, 2-green, 0-red
11	t_ph_start	Numeric	Seconds after start of phase (yellow and green are same phase)
12	t_ph_end	Numeric	Seconds to end of phase (yellow and green are same phase)
13	t_occ	Numeric	In seconds
14	speed	Numeric	In mph
15	num_sens	Integer	Number of sensors
16	t_detect_s1_epoch	Numeric	Detect time at sensor 1
17	t_detect_s1_local	Text	Col 16 in local time
18	t_undetect_s1_epoch	Numeric	Undetect time at sensor 1
19	t_undetect_s1_local	Text	Col 18 in local time
20	t_detect_s2_epoch	Numeric	Detect time at sensor 2 (if sensor doesnt exist, -1)
21	t_detect_s2_local	Text	Col 20 in local time (if sensor doesnt exist, -1)
22	t_undetect_s2_epoch	Numeric	Undetect time at sensor 2 (if sensor doesnt exist, -1)
23	t_undetect_s2_local	Text	Col 22 in local time (if sensor doesnt exist, -1)

Table B.7: Lane Events Table

Example Query Code

```
import sqlite3
from sqlite3 import Error
import pandas as pd
def create_connection(db_file):
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)
        return None
database = "highres.db"
conn = create_connection(database)
with conn:
    """
    You can edit the date range you want to extract
    from the database. Lookup the equivalent Epoch
    time, and then insert it in the "WHERE
    t_detect_s1_epoch" clause.

    The data range used in this example is:
    Date range: 12 Feb 2017 to 26 Feb 2017 (2 weeks)
                1486875600   1488085200+86399=1488171599

    """
    sql_command = '''SELECT t_detect_s1_local,
    intersection, legnum, lanenum, lanetype, t_occ,
    speed, event_type, sp, sp_valid, t_ph_start,
    t_ph_end
    FROM laneEvents
    WHERE t_detect_s1_epoch BETWEEN 1486875600
    and 1488171599; ''' ##

    print("Connecting to database..")
    cur = conn.cursor()
    print("Fetching data..")
    cur.execute(sql_command)
    rows = cur.fetchall() ## type = list

    print("Making a dataframe..")
    df = pd.DataFrame.from_records(rows,
```

```
    columns=['t','intersection', 'legnum',
    'lanenum', 'lanetype', 't_occ', 'speed',
    'sensorType', 'sp', 'sp_valid', 't_ph_start',
    't_ph_end'])

    print("Converting types..")
    df['t_occ'] = df['t_occ'].apply(lambda x: float(x))
    df['legnum'] = df['legnum'].apply(lambda x: int(x))
    df['lanenum'] = df['lanenum'].apply(lambda x: int(x))
    df['lanetype'] = df['lanetype'].apply(lambda x: int(x))
    df['speed'] = df['speed'].apply(lambda x: float(x))
    df['t'] = df['t'].apply(lambda x: str(x))
    df['sensorType'] = df['sensorType'].apply(lambda x: str(x))
    df['t_ph_start'] = df['t_ph_start'].apply(lambda x: float(x))
    df['t_ph_end'] = df['t_ph_end'].apply(lambda x: float(x))
    """
    Here we want to cast integers to uint8 to
    make the file smaller.
    """
    df['intersection'] = df['intersection'].astype('uint8')
    df['legnum'] = df['legnum'].astype('uint8')
    df['lanenum'] = df['lanenum'].astype('uint8')
    df['lanetype'] = df['lanetype'].astype('uint8')
    df['sp'] = df['sp'].astype('uint8')
    df['sp_valid'] = df['sp_valid'].astype('uint8')

    print("Now remember to save..")
    outputFileName = "Feb2Weeks.h5"
    outputName = "df"
    df.to_hdf(outputFileName,outputName,
    mode='w',table=True)
    print("Done!")

    """
    Now, you can load the processed file
    for use in other scripts by using this code in
    another script.
    """
    # outputFileName = "Feb2Weeks.h5"
    # outputName = "df"
    #df = pd.read_hdf(outputFileName,outputName)
```

REFERENCES

[1] A. Stathopoulos and M. G. Karlaftis, "A multivariate state space approach for urban traffic flow modeling and prediction," *Transportation Research Part C: Emerging Technologies*, vol. 11, no. 2, pp. 121 –135, 2003.

[2] Y. Kamarianakis, W. Shen, and L. Wynter, "Realtime road traffic forecasting using regimeswitching spacetime models and adaptive lasso," *Applied Stochastic Models in Business and Industry*, vol. 28, no. 4, pp. 297–315, eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/asmb.1937`.

[3] W. Min and L. Wynter, "Real-time road traffic prediction with spatio-temporal correlations," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 4, pp. 606 –616, 2011.

[4] Y. Kamarianakis, H. O. Gao, and P. Prastacos, "Characterizing regimes in daily cycles of urban traffic using smooth-transition regressions," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 5, pp. 821 –840, 2010, Applications of Advanced Technologies in Transportation: Selected papers from the 10th AATT Conference.

[5] I. M. Marko Budišić Ryan M. Mohr, "Applied koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, 2012.

[6] B. O. Koopman, "Hamiltonian systems and transformation in hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, 1931. eprint: `http://www.pnas.org/content/17/5/315.full.pdf`.

[7] Y. Susuki, I. Mezic, F. Raak, and T. Hikihara, "Applied koopman operator theory for power systems technology," *Nonlinear Theory and Its Applications, IEICE*, vol. 7, no. 4, pp. 430–459, 2016.

[8] C. W. Rowley, I. Mezic, S. Bagheri, P. Schlatter, and D. S. Henningson, "Spectral analysis of nonlinear flows," *Journal of Fluid Mechanics*, vol. 641, pp. 115–127, 2009.

[9] H. Arbabi and I. Mezić, "Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator," *SIAM Journal on Applied Dynamical Systems*, vol. 16, no. 4, pp. 2096–2126, 2017. eprint: `https://doi.org/10.1137/17M1125236`.

[10] I. Mezić, "Analysis of fluid flows via spectral properties of the koopman operator," *Annual Review of Fluid Mechanics*, vol. 45, no. 1, pp. 357–378, 2013. eprint: https://doi.org/10.1146/annurev-fluid-011212-140652.

[11] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, 2010.

[12] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: Theory and applications," *Journal of Computational Dynamics*, vol. 1, no. 2, pp. 391–421, 2014.

[13] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, Dec. 2015.

[14] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis, "A kernel-based method for data-driven koopman spectral analysis," *Journal of Computational Dynamics*, vol. 2, no. 2, pp. 247–265, 2015.

[15] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," *Nonlinear Dynamics*, vol. 41, no. 1, pp. 309–325, Aug. 2005.

[16] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PLOS ONE*, vol. 11, no. 2, pp. 1–19, Feb. 2016.

[17] J. L. Proctor and P. A. Eckhoff, "Discovering dynamic patterns from infectious disease data using dynamic mode decomposition," *International Health*, vol. 7, no. 2, pp. 139–145, Mar. 2015.

[18] P. Schmid and J. Sesterhenn, "Dynamic Mode Decomposition of numerical and experimental data," in *APS Division of Fluid Dynamics Meeting Abstracts*, Nov. 2008, MR.007.

[19] P. V. Overschee and B. D. Moor, *Subspace identification for linear systems: Theory, implementation, applications*. Kluwer, 1996.

[20] B. Boots, G. J. Gordon, and S. M. Siddiqi, "A constraint generation approach to learning stable linear dynamical systems," in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds., Curran Associates, Inc., 2008, pp. 1329–1336.

[21] P. Rapp, T. Schmah, and A. Mees, "Models of knowing and the investigation of dynamical systems," *Physica D: Nonlinear Phenomena*, vol. 132, no. 1, pp. 133 – 149, 1999.

[22] A. Venkatraman, B. Boots, M. Hebert, and J. A. Bagnell, "Data as demonstrator with applications to system identification," NIPS, NIPS Autonomously Learning Robots Workshop, 2014.

[23] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," 29th, Association for the Advancement of Artificial Intelligence, AAAI Conference on Artificial Intelligence, 2015.

[24] W. Huang, L. Cao, F. Sun, D. Zhao, H. Liu, and S. Yu, "Learning stable linear dynamical systems with the weighted least square method," 25th, International Joint Conference on Artificial Intelligence, 2016.

[25] N. J. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic mode decomposition: Data-driven modeling of complex systems*. Society for Industrial and Applied Mathematics (SIAM), 2016.

[26] J. L. L. Gal Berkooz Philip Holmes, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual Review of Fluid Mechanics*, vol. 25, pp. 539–75, 1993.

[27] M. J. Lighthill, *An introduction to fourier analysis and generalised functions*. Cambridge University Press, Jan. 1958.

[28] I. Jolliffe, "Principal component analysis," in *Encyclopedia of Statistics in Behavioral Science*. American Cancer Society, 2005, ISBN: 9780470013199. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470013192.bsa501.

[29] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, 2016. eprint: https://doi.org/10.1137/15M1013857.

[30] F. Raak, Y. Susuki, I. Mezić, and T. Hikihara, "On koopman and dynamic mode decompositions for application to dynamic data with low spatial dimension," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 6485–6491.

[31] Y. Susuki and I. Mezić, "Nonlinear koopman modes and power system stability assessment without models," *IEEE Transactions on Power Systems*, vol. 29, no. 2, pp. 899–907, Mar. 2014.

[32] F. Raak, Y. Susuki, and T. Hikihara, "Data-driven partitioning of power networks via koopman mode analysis," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 2799–2808, Jul. 2016.

[33] S. Brunton, J Nathan Kutz, X. Fu, and J. Grosek, *Handbook of robust low-rank and sparse matrix decomposition: Applications in image and video processing*. CRC Press, Jun. 2016, ch. 19, ISBN: 9781498724623.

[34] J. Mann and J. N. Kutz, "Dynamic mode decomposition for financial trading strategies," *Quantitative Finance*, vol. 16, no. 11, pp. 1643–1655, 2016. eprint: https://doi.org/10.1080/14697688.2016.1170194.

[35] Y. Susuki and I. Mezic, "Nonlinear koopman modes and coherency identification of coupled swing dynamics," *IEEE Transactions on Power Systems*, vol. 26, no. 4, pp. 1894–1904, Nov. 2011.

[36] S. L. Brunton, J. L. Proctor, J. H. Tu, and J. N. Kutz, "Compressed sensing and dynamic mode decomposition," *Journal of Computational Dynamics*, vol. 2, no. 2, pp. 165–191, 2015.

[37] N. Takeishi, Y. Kawahara, Y. Tabei, and T. Yairi, "Bayesian dynamic mode decomposition," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 2814–2821.

[38] C. Liu, B. Huang, M. Zhao, S. Sarkar, U. Vaidya, and A. Sharma, "Data driven exploration of traffic network system dynamics using high resolution probe data," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 7629–7634.

[39] Z. Amini, R. Pedarsani, A. Skabardonis, and P. Varaiya, "Queue-length estimation using real-time traffic data," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2016, pp. 1476–1481.

[40] S. A. Fayazi, A. Vahidi, G. Mahler, and A. Winckler, "Traffic signal phase and timing estimation from low-frequency transit bus data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 19–28, Feb. 2015.

[41] G. Mahler and A. Vahidi, "Reducing idling at red lights based on probabilistic prediction of traffic signal timings," in *2012 American Control Conference (ACC)*, Jun. 2012, pp. 6557–6562.

[42] *Traffic signal timing manual, chapter 4*, U.S. Department of Transportation, Aug. 2017.

[43] Z. Bar-Yehuda, *Plot google map*, MathWorks File Exchange, Apr. 2018.

[44] *Traffic signal timing manual, chapter 7*, U.S. Department of Transportation, Aug. 2017.

[45] *Traffic signal timing manual, chapter 2*, U.S. Department of Transportation, Aug. 2017.

[46] *Maryland open data portal*, Maryland.gov, Aug. 2017.

[47] A. Blum and Y. Mansour, "Learning, regret minimization, and equilibria," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, Eds. Cambridge University Press, 2007, 79102.